

UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR



Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

SISTEMA DE GESTIÓN DE CLUBS DE GOLF

Autor: Roberto Blázquez Pardo

Tutor: José Antonio Clavijo Blázquez

Ponente: Francisco de Borja Rodríguez Ortiz

Junio 2017

SISTEMA DE GESTIÓN DE CLUBS DE GOLF

Autor: Roberto Blázquez Pardo
Tutor: José Antonio Clavijo Blázquez
Ponente: Francisco de Borja Rodríguez Ortiz

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2017

Resumen

Resumen

Este proyecto consiste en el análisis, diseño y el desarrollo de una fase de un sistema de gestión de clubs de golf. El objetivo es agilizar los procesos, que a día de hoy realizan con otra herramienta o manualmente dentro de un club, para que Delonia pueda distribuir su acceso, entre los distintos clubs, además de poder controlar los datos de los clubs dentro de la aplicación.

El cliente es una empresa encargada de distribuir sus productos entre los distintos clubs de golf, para ayudarles a administrar y mejorar las tareas que tienen que realizar. En este momento disponen de una aplicación web, donde los usuarios principales son los clientes del club, debido a que pueden gestionar y confirmar reservas de los Green Fees y torneos con ella. La aplicación que nos han encargado está dirigida a los empleados, ya que serán los usuarios principales, con el objetivo de aumentar su rendimiento.

El sistema está subdividido en varios subsistemas: de gestión de usuarios, clientes y socios, de campos y productos, reservas, torneos, profesores, facturas, clubs y subsistema para la comunicación con la base de datos.

A diferencia de otras aplicaciones, la nuestra intenta englobar casi toda la gestión de un club de golf y el objetivo de realizar, en el menor tiempo posible, las tareas más comunes de los empleados de un club.

Durante años, la elección por parte de los desarrolladores para elaborar una herramienta informática, han sido las aplicaciones de escritorio, pero durante los últimos años, se ha ido avanzando hacia las aplicaciones web, debido a la versatilidad que proporcionan los nuevos lenguajes de programación y la mejora de la conectividad.

La herramienta está basada en una aplicación web desarrollada en HTML5, CSS3 y JavaScript 6 en el frontend, y en Java y PostgreSQL en el backend.

Palabras Clave

Golf, Aplicación Web, Sistema ERP, Gestión, Clubs, Delonia, SCRUM.

Abstract

This project consists of the analysis, design and development of one of the phases of golf club management system. The objective is to speed up the processes, that nowadays are done with another tool or manually in the club, so that Delonia can distribute its access, between the different clubs, besides being able to control the data of the clubs data within the application.

The client is a company in charge of distributing its software between different golf clubs, to help them manage and improve the tasks they have to perform. At this moment, the client has a web application, where the main users are the clients of the club, from where they can manage and confirm reservations of Green Fees and the tournaments. The application we have to develop is addressed to the employees, them being the main users, with the objective of increasing their performance.

The system is divided into several subsystems: user management, client and partner, field and product, reservations, tournaments, teachers, invoices, clubs and subsystem for communication with the database.

Unlike other applications, ours tries to encompass almost all the tasks needed for the golf club management with the objective of performing in the shortest possible time the most common tasks of employees of the club.

During all these years, the choice of the developers has been to develop a desktop application, but during the last years, progress has been made towards web applications, because of the versatility provided by the new programming languages and improved connectivity to the web.

The frontend of the web application is developed in HTML5, CSS3 y JavaScript 6, and the backend in Java and PostgreSQL.

Key words

Golf, Web Application, ERP System, Management, Clubs, Delonia, SCRUM.

Agradecimientos

Quiero agradecer a todos los profesores que he tenido durante estos cuatro años de carrera, por enseñarme los conocimientos necesarios para poder realizar mi proyecto de TFG.

También a mi tutor en la empresa Delonia, José Antonio, por la ayuda prestada durante el proyecto, extensiva al arquitecto, Juan Luis, por todas las explicaciones que me ha dado durante mi periodo en la empresa.

También quisiera mencionar a mi ponente, Francisco, por ayudarme con las correcciones en la memoria.

Por último, quisiera agradecer a mi familia por aguantarme durante todos estos años de enseñanza, y permitir que pudiera seguir estudiando.

Índice general

Índice de Figuras	x
Índice de Tablas	xii
1. Introducción	1
1.1. Marco del proyecto	1
1.2. Motivación	1
1.3. Objetivo	2
1.4. Estructura del documento	2
2. Estado del arte	3
2.1. Historia del golf	3
2.2. Gestión de un club de golf	3
2.3. Análisis de sistemas de gestión de clubs de golf	4
2.3.1. Herramientas estándar para la gestión de clubs de golf	4
2.3.2. Herramientas a medida para la gestión de clubs de golf	5
2.4. Herramientas de desarrollo web	5
2.4.1. HTML, CSS y JavaScript	5
2.4.2. Java, ASP.NET y PHP	7
2.4.3. Base de Datos	8
3. Definición del proyecto	9
3.1. Metodología	9
3.1.1. SCRUM	9
3.2. Definición del proyecto	11
3.3. Alcance del proyecto	12
3.4. Subsistemas del proyecto	12
4. Análisis	15
4.1. Usuarios de la aplicación	15
4.2. Análisis de requisitos	16

4.2.1. Requisitos funcionales	16
4.2.2. Requisitos no funcionales	20
5. Diseño	21
5.1. Diagramas de flujo	21
5.2. Maquetas	21
5.3. Base de datos	21
5.3.1. Proceso	22
5.3.2. Normalización	22
5.3.3. Desnormalización	24
5.3.4. Herencia	25
6. Desarrollo	27
6.1. Generador de datos	27
6.2. Tecnologías utilizadas	29
6.2.1. NetBeans IDE 8.2	29
6.2.2. ReactJS	29
6.2.3. Uaithne	30
6.2.4. Visual Paradigm	30
6.2.5. pgAdmin	31
6.3. Arquitectura de la aplicación	31
6.3.1. Apache Tomcat	31
6.3.2. Comunicación cliente-servidor	31
6.4. Base de datos	32
7. Pruebas Realizadas y Resultados	33
7.1. Base de datos	33
7.2. Pruebas de verificación	34
7.2.1. Pruebas backend	34
8. Conclusiones y trabajo futuro	35
Glosario de acrónimos	37
Bibliografía	37
A. Diagramas de flujo	41
A.0.1. Proceso de nueva reserva	41
A.0.2. Proceso para hacer una reserva efectiva	43

A.0.3. Proceso de cancelación de reserva	43
A.0.4. Proceso de nueva membresía	44
B. Maquetas	45
B.0.1. Proceso de nueva reserva	45
C. Generador de datos	49
C.1. Código	49
C.2. Ejemplo	60

Índice de Figuras

2.1. Logo MasterGolf	4
2.2. Interfaz CONCEPT GOLF	5
2.3. Logo HTML5, CSS3 y JavaScript 6	6
3.1. Diagrama proceso SCRUM	10
3.2. Tablero SCRUM	11
3.3. Diagrama de dependencia de los subsistemas	13
5.1. Desnormalización BD	25
5.2. Herencia BD	25
6.1. Diagrama de clases del generador de datos	29
6.2. Logo ReactJS	30
6.3. Arquitectura web	31
6.4. Petición Json	32
6.5. Respuesta Json	32
7.1. Consulta sin desnormalizar	34
7.2. Consulta con desnormalización	34
A.1. Diagrama de flujo nueva reserva	42
A.2. Diagrama de flujo hacer efectiva una reserva	43
A.3. Diagrama de flujo cancelar reserva	43
A.4. Diagrama de flujo nueva membresía	44
B.1. Primera etapa proceso de nueva reserva	45
B.2. Segunda etapa proceso de nueva reserva	46
B.3. Tercera etapa sin elección de profesor en proceso de nueva reserva	47
B.4. Tercera etapa con elección de profesor en proceso de nueva reserva	47
C.1. Tabla Persona	61

Índice de Tablas

4.1. Roles de Usuario	15
5.1. Tabla de membresía sin normalizar	22
5.2. Tabla de membresías en primera forma normal	23
5.3. Tabla de membresías en segunda forma normal	23
5.4. Tabla de miembros de las membresías	23
5.5. Tabla de membresías en tercera forma normal	24
5.6. Tabla de tipos de membresía	24
6.1. Comparación de Angular 2 y ReactJS	30

1

Introducción

1.1. Marco del proyecto

Este Trabajo de Fin de Grado (TFG) consiste en la realización del análisis y diseño completo de un **Sistema ERP de Gestión de Clubs de Golf**, y del desarrollo de la primera fase para Delonia, que luego a su vez dará acceso al sistema a los distintos clubs.

El sistema será una herramienta encargada de gestionar el inventario, los socios/clientes, los profesores, los recorridos, las reservas y los torneos de cada club de golf. Con el objetivo de mejorar sistema manual usado actualmente para la mayoría de estas gestiones, y aumentando el rendimiento de los trabajadores del club, encargados de estas tareas.

1.2. Motivación

La gestión de un club de golf es un proceso costoso, incluso existiendo cursos para gestión de golf, por ejemplo el portal web [golfindustria\[1\]](#), por ello, si no tienes las herramientas adecuadas, los tiempos de las tareas y el espacio necesario para almacenar los datos se multiplican considerablemente, ya que en un disco duro, podemos almacenar información de cientos e incluso miles de archivadores.

Otro problema al que nos enfrentamos es a la realización de estadísticas, ya sea por parte del cliente o por algún club que use el sistema. Es un proceso de días o incluso semanas, dependiendo de la cantidad de datos si lo realizamos a mano, por lo que es imprescindible tener una herramienta que sea capaz de generar estadísticas en segundos. Por ejemplo si un administrador global desea ver que club le ha generado más ingresos, si la estadística la realiza a mano o en un Excel, serían una gran cantidad de datos que introducir y cálculos que realizar, en cambio, con nuestra aplicación, solo se realizara un clic en un botón para que la estadística se genere.

Los procesos que más se repiten en un club de golf es la creación de una reserva por parte de un empleado del club y hacer esta efectiva en el momento en que el cliente ocupa el campo de golf. Debido a la frecuencia de estos dos procesos, son los que más tenemos que analizar para que se puedan realizar en el menor tiempo posible. Sin la herramienta adecuada el proceso de

reserva sería muy complicado, ya que un club tendrá decenas de reservas en un mismo día, y con una buena herramienta podrá ver lo huecos libres en apenas dos clics.

1.3. Objetivo

El principal objetivo es el análisis, diseño y desarrollo de la primera fase de un **Sistema de Gestión de Clubs de Golf**, que corresponderá al desarrollo de la base de datos, una implementación inicial del backend y la arquitectura de comunicaciones con el frontend. El sistema se implantará en producción por fases, según se vayan desarrollando los subsistemas.

El sistema debe de ser capaz de realizar la gestión del inventario, los socios/clientes, los profesores, los recorridos, las reservas y los torneos, teniendo siempre en cuenta la creación de una solución a los problemas anteriores. Otro objetivo fundamental es facilitar que tiene que tener una interfaz sencilla y cómoda para el usuario, debido a que el empleado de un club va a pasar gran parte de su tiempo en el trabajo utilizando el sistema.

El proyecto lo dividiremos en subobjetivos.

1. Realización del análisis de la aplicación y su posterior validación por parte del arquitecto, en este caso sería el Arquitecto de la empresa.
2. Realizar el diseño de la aplicación y obtener una primera versión en *papel* de la aplicación validada por el cliente.
3. Creación de la base de datos, realizando la población de datos con un generador de datos, para a continuación comprobar el correcto funcionamiento de la misma.
4. Realizar una implementación inicial del backend de la aplicación, conectado con la base de datos y realizando sus respectivas pruebas. También el desarrollo de la arquitectura de comunicaciones con el frontend.

1.4. Estructura del documento

En el segundo capítulo se realiza un análisis del estado del arte, donde veremos la historia del golf, de que herramientas principales se dispone para construir una aplicación web y un análisis de herramientas parecidas a la desarrollada.

En el tercer capítulo se explica la metodología utilizada en el proyecto y una descripción del mismo.

En el cuarto capítulo se realiza el análisis de la aplicación, que nos ayudara a entender mejor el sistema para luego realizar el diseño.

En el quinto capítulo se realiza el diseño de la aplicación que será indispensable para posteriormente poder desarrollar el proyecto.

En el sexto capítulo veremos el desarrollo y las herramientas que hemos utilizado.

En el séptimo capítulo veremos las pruebas que hemos realizado para comprobar su calidad y funcionamiento.

Por último, en el octavo capítulo, veremos las conclusiones que hemos sacado durante la elaboración del proyecto y el trabajo futuro que se realizará a continuación de lo realizado en este TFG.

2

Estado del arte

2.1. Historia del golf

En la actualidad aún no se han descubierto los orígenes del golf como lo conocemos hoy en día. En los **Países Bajos**, en 1297, los holandeses jugaban a un juego con un palo y una bola de cuero, llamado *Colf*, donde el objetivo era conseguir dar a un blanco en el menor número de golpes posibles. Aunque unos años antes, en **Francia** está recogido el *Chole* hacía 1200, y unos años más tarde, en 1450, el diario de Adelaida de Saboya posee una representación de un golfista. En **Bruselas** se prohibió el *Colven* en 1360, y sobre la misma época, se descubrió que en la catedral de **Gloucester** existe una vidriera de 1340 que muestra un golfista.

A pesar de todas las referencias, el golf moderno es considerado un invento de **Escocia**, como se recoge en dos actas del Parlamento de Escocia en el **siglo XV**, donde se expandió a Inglaterra y a continuación al resto del mundo.

El club de golf más antiguo del mundo es el de la **Honourable Company of Edinburgh Golfers** con sede en Muirfield, donde encontramos el primer reglamento escrito, junto al diseño del campo de golf de 18 hoyos. Los primeros torneos formales y competiciones entre ciudades se registran en Escocia [2].

2.2. Gestión de un club de golf

La gestión de un club de golf la podemos dividir en cinco categorías principales.

1. **Gestión de empleados.** Categoría que se encarga de que los empleados realicen bien su trabajo y rindan al máximo.
2. **Apartado económico.** Categoría encargada de las finanzas del club.
3. **Gestión de instalaciones.** Como su nombre indica, esta categoría se encarga de la administración y mantenimiento de las instalaciones.
4. **Gestión deportiva y operaciones.** Categoría que se encarga del inventario y dar servicio a los clientes.

5. **Marketing.** Área encargada de atraer clientes al club a través de la publicidad.

2.3. Análisis de sistemas de gestión de clubs de golf

2.3.1. Herramientas estándar para la gestión de clubs de golf

MasterGolf

Hay que mencionar que MasterGolf no es una sola herramienta, si no, un conjunto de ellas teniendo cada una su función específica: Repwin para la gestión de reservas y salidas del campo, Golfwin para la gestión de socios, Compwin para la gestión de competiciones del club, Prowin para el sorteo de salidas al campo y Golfshow nos permite ver el vídeo en streaming. En conjunto es una aplicación que abarca la mayoría de actividades que se les puede presentar a los empleados de un campo de golf, pero si nosotros vemos la interfaz, la sensación es de una aplicación antigua y no muy agradable a la vista. También podemos apreciar, siendo un usuario que la ve como primera vez, que es una herramienta poco intuitiva debido al exceso de información que se muestra en ella [3]. A diferencia, nuestra aplicación, engloba todas las gestiones en una misma herramienta y la cantidad de información en una misma pantalla no es excesiva.



Figura 2.1: Logo MasterGolf. Fuente: <https://www.golfspain-mastergolf.com/>

CONCEPT GOLF

CONCEPT GOLF es una aplicación dirigida únicamente a la gestión de reservas y de socios. Observando la aplicación podemos ver que es un herramienta que cubre casi todas las funciones con respecto a las reservas y los socios, pero si nos fijamos bien en su interfaz, podemos ver una serie de colores que no dan la sensación de estar gestionando campos de golf, aparte de que no están bien elegidos para la lectura de los textos, este efecto se puede apreciar perfectamente en la figura 2.2 donde han elegido el color rojo para algunos recuadros de la tabla que pone en gran dificultad la lectura del texto [4]. En cambio, nosotros hemos elegido unos colores que nos indican que estamos realizando tareas relacionadas con un campo de golf, también hemos elegido unos colores de fondo que permitan una lectura fácil de la misma.

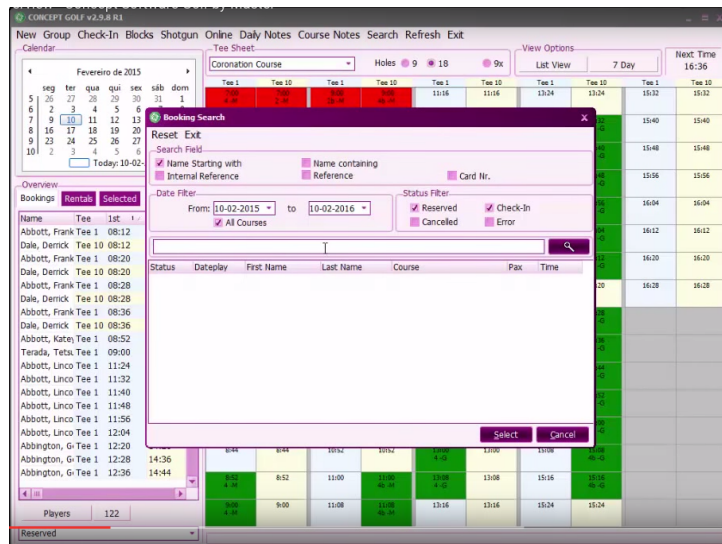


Figura 2.2: Interfaz CONCEPT GOLF. Fuente: <http://www.mastelhospitality.com/golf-software-solutions/>

2.3.2. Herramientas a medida para la gestión de clubs de golf

Las herramientas a medidas son sistemas que han sido creados bajo demanda de una empresa para mejorar el rendimiento de sus empleados en las actividades que realizan, por ello, la aplicación debe adaptarse lo mejor posible al proceso de trabajo de la empresa.

Una de las principales desventajas de una herramienta a medida es que son mucho más caras que las estándar, debido a que ha sido diseñada únicamente para una empresa con una serie de especificaciones propias y elegidas por la misma. A pesar de ser más caras, esta diferencia de precio es justificable debido a que con una herramienta a medida el rendimiento es mucho mayor, por lo tanto el beneficio al final de mes para la empresa también será mayor.

Por último, debemos mencionar que si un cliente encarga una herramienta a medida, es muy probable que haya restringido el uso de esta para su empresa, lo que garantiza al contratante, el uso privativo de la aplicación.

2.4. Herramientas de desarrollo web

A continuación realizamos un análisis de las distintas herramientas básicas y más populares que podemos elegir a la hora de desarrollar una aplicación web.

2.4.1. HTML, CSS y JavaScript

HTML, CSS y JavaScript es el conjunto de lenguajes utilizados para el desarrollo de las aplicaciones Web en la parte del cliente. En este periodo de tiempo podemos decir que es impensable usar uno de ellos sin acabar usando los otros dos, ya que se complementan entre ellos.

HTML

Conocido como **HyperText Markup Language**, es un lenguaje que salió de las etiquetas SGML. Estas etiquetas interpretadas por el navegador son las encargadas de dar la estructura



Figura 2.3: Logo HTML5, CSS3 y JavaScript 6.

Fuente: <https://i.ytimg.com/vi/8sEXj0U5X8g/maxresdefault.jpg>

y de añadir elementos a la aplicación web.

Con la última versión, HTML5, podemos decir que el lenguaje ha avanzado a grandes pasos, sobretodo en la creación de nuevas etiquetas para la estructura del documento y para la reproducción de vídeo y audio [5].

CSS

Conocido como **Cascading Style Sheets** es el lenguaje utilizado para describir la presentación de aplicaciones web en HTML y XML. Es el encargado de decidir cómo se debe pintar el elemento. Podemos decir que utilizamos HTML5 para elegir el “QUE” pintamos y CSS para el “COMO”.

La última versión CSS3, que vino de la mano con HTML5, ha evolucionado a un gran nivel, destacando las nuevas formas con que crear efectos por su sencillez con respecto a su antigua versión, como puede ser el estilo de las sombras, también cabe destacar la incorporación de nuevos layouts, como las cajas flexibles [6].

Existen dos herramientas que nos ayudan a programar más rápido los estilos CSS. **SASS** destaca por el uso de variables, anidamiento de estilos y mixins, también posee varias funciones para el manejo de colores y otros valores permitiendo el uso de elementos básicos de programación como las directrices de control y las librerías. Y por otro lado tenemos **LESS**, mucho más amigable que SASS debido a su gran parecido a la hora de programar con CSS, a pesar de no ser tan potente, también dispone de variables, anidamientos, operadores, mixins y funciones. La principal diferencia entre LESS y otros pre-procesadores CSS es que permite la compilación en tiempo real [7].

JavaScript

Es un lenguaje de programación interpretado. Podemos decir que es, por excelencia, el lenguaje para realizar aplicaciones web dinámicas. Es el más común de todos, por ello se han desarrollado multitud de frameworks con él, de los cuales podemos destacar algunos como AngularJS o ReactJS para el lado del cliente (donde es más común la presencia de JavaScript), aunque también se ha llevado al lado del servidor, como podemos ver en NodeJS.

No podemos encasillarlo en una categoría debido a que engloba la mayoría de ellas. Como ya hemos visto, se puede utilizar para el cliente o el servidor, pero no solo se queda ahí, también es un lenguaje que es utilizado para el desarrollo de videojuegos, como puede ser en Unity 5 [8].

2.4.2. Java, ASP.NET y PHP

Cualquier persona relacionada con el desarrollo web, sabe que Java, ASP.NET y PHP son las tres herramientas más usadas por los desarrolladores en la parte del servidor para una aplicación web, aunque JavaScript también está extendiendo en este ámbito.

Java

Es un lenguaje de programación orientado a objetos, que deriva de los lenguajes de programación C y C++, pero aun nivel más alto.

Es muy versátil a pesar de que la mayoría de los desarrolladores lo utilizan para implementar el servidor de una aplicación web, también es posible crear aplicaciones de escritorio con Swing o implementar interfaces de usuario en el servidor con frameworks, como Vaadin, GWT o GXT.

En nuestro caso la hemos elegido como lenguaje para desarrollar el servidor por su gran cantidad de frameworks, que nos ayudan y reducen el tiempo del proyecto, a diferencia de los otros lenguajes de programación y al fin y al cabo es lo que nos interesa a la hora de hacer un análisis previo, *ahorrar*.

ASP.NET

Es un entorno para el desarrollo de aplicaciones web y comercializado por **Microsoft**. Permite desarrollar aplicaciones que se beneficien del **Common Language Runtime**, y escribir código en cualquier lenguaje aceptado por **.NET Framework**, como puede ser C#. Esta tecnología es la sucesora de **Active Server Pages** (ASP).

Su objetivo es el desarrollo de aplicaciones web empresariales con el mínimo código posible. ASP.NET lleva incorporado una estructura de seguridad, tema imprescindible en la mayoría de las aplicaciones web empresariales.

Aunque disponga de una estructura de seguridad, que es un punto muy a favor, el soporte mayoritario es para Microsoft Windows, el cual no es nada eficiente para alojar un servidor.

PHP

Conocido como **Hypertext Preprocessor**, es un lenguaje de código abierto, destacado en el mundo del desarrollo web y que puede ser incrustado en HTML [9].

El código PHP se ejecuta en el servidor y es este el que envía la página HTML al cliente, por lo que todo el trabajo se concentra en el servidor. Fue uno de los primeros lenguajes de programación capaz de incorporar su código en los documentos HTML, en lugar de llamar a un archivo externo.

Fue creado originalmente por **Rasmus Lerdorf** en el año 1995, aunque su implementación principal es realizada en la actualidad por **The PHP Group** [10].

En nuestro caso hemos descartado PHP porque todo el trabajo se delega al servidor, y nosotros entendemos que no aprovechar el procesador del cliente y sobrecargar el servidor es algo que no nos podemos permitir.

2.4.3. Base de Datos

Podemos definir una base de datos como el conjunto de información perteneciente a un mismo contexto y almacenada sistemáticamente para su posterior uso.

Existen multitud de sistemas de gestión de Bases de Datos (BBDD), pero destacaría tres por encima de las demás por su gran desarrollo y popularidad: **Oracle**, **Microsoft SQL Server** y **PostgreSQL**.

Oracle

Base de datos de tipo **objeto-relacional**, comercial y desarrollado por **Oracle Corporation**. Surgió por primera vez en **1977** bajo el nombre de **SDL** (Software Development Laboratories). En **1979** cambió su nombre por **Relational Software, Inc** (RSI).

Fue uno de los primeros sistemas de gestión de BBDD desarrollados, por ello atrajo la gran mayoría del mercado comercial hasta la aparición de **Microsoft SQL Server**. En las últimas versiones Oracle ha sido certificado para poder trabajar bajo GNU/Linux [11].

Hemos descartado Oracle, a pesar de ser una de las mejores, por ser comercial y podemos tener resultados similares con PostgreSQL totalmente gratis como veremos después.

Microsoft SQL Server

Sistema de gestión de bases de datos basado en el modelo relacional, comercial y desarrollado por **Microsoft**. El lenguaje utilizado es **Transact-SQL** (TSQL), una implementación del estándar ANSI del lenguaje SQL. A pesar de su uso limitado para el sistema operativo **Windows** de Microsoft, es una de las bases de datos mejor colocados en el mercado.

Se compone de 5 ediciones, **Enterprise** como la edición más completa, **Developer** que está diseñada para utilizarse en desarrollo y no en producción, **Standard** para servidores inferiores, **Express** como la única edición gratuita y **SQL Azure** para tener un servidor en la nube pagando una mensualidad [12].

SQL Server se ha descartado por el mismo motivo que ASP.NET, no queremos alojar nuestra aplicación en un servidor Microsoft Windows.

PostgreSQL

Sistema de gestión de bases de datos objeto-relacional, distribuido bajo la licencia de **BSD** y de código abierto. Fue desarrollado hace más de 16 años. A pesar de ser totalmente gratuito, no tiene nada que enviar al resto de bases de datos comerciales como las mencionadas anteriormente.

Está basado en el modelo cliente-servidor. Una de sus características más significativas es el uso de multiprocesos en vez de multihilos, para poder mantener en funcionamiento una tarea en caso de que otra sufra un fallo [13].

En nuestro caso hemos elegido PostgreSQL debido a que es una de las bases de datos que soportan una gran cantidad de información, y su utilización es completamente gratuita.

3

Definición del proyecto

3.1. Metodología

Definida por la **RAE** como el “*Conjunto de métodos que se siguen en una investigación científica o en una exposición doctrinal*” [14]. Conociendo su significado e investigando un poco podemos llegar a la conclusión de que la **Metodología** es el conjunto de métodos, técnicas y herramientas utilizadas con el objetivo de realizar una tarea.

Si lo aplicamos al desarrollo software, podemos entenderlo como la guía que debemos seguir a lo largo de todas las etapas de nuestro proyecto (análisis, diseño, desarrollo, pruebas y mantenimiento) con el objetivo de cumplir con los requisitos del cliente y entregarle un producto de calidad en el menor tiempo posible. En caso de no utilizar una metodología, podemos estar casi seguros de que el proyecto acabará en un desastre debido a la falta de organización por las distintas partes del equipo.

Entre las distintas metodologías ágiles analizadas, se ha decidido utilizar **SCRUM** debido a que la hemos considerado como la que más valor puede aportar a nuestro proyecto, siendo iterativa, permitiendo involucrar al cliente en el desarrollo y llevar las tareas al día.

3.1.1. SCRUM

Es una metodología de desarrollo incremental, y fomenta el trabajo en equipo para conseguir el objetivo [15].

Proceso

Se ejecuta en bloques temporales y fijos, donde a cada bloque se le denomina **Sprint** o iteración. Cada uno de ellos debe dar un resultado completo para ser entregado al cliente. En la figura 3.1 podemos ver gráficamente el proceso SCRUM.

- **Product Backlog.** Es una lista de las funcionalidades del producto ordenada según la prioridad, elaborada por el **Product Owner**, el cual se explicará más adelante.

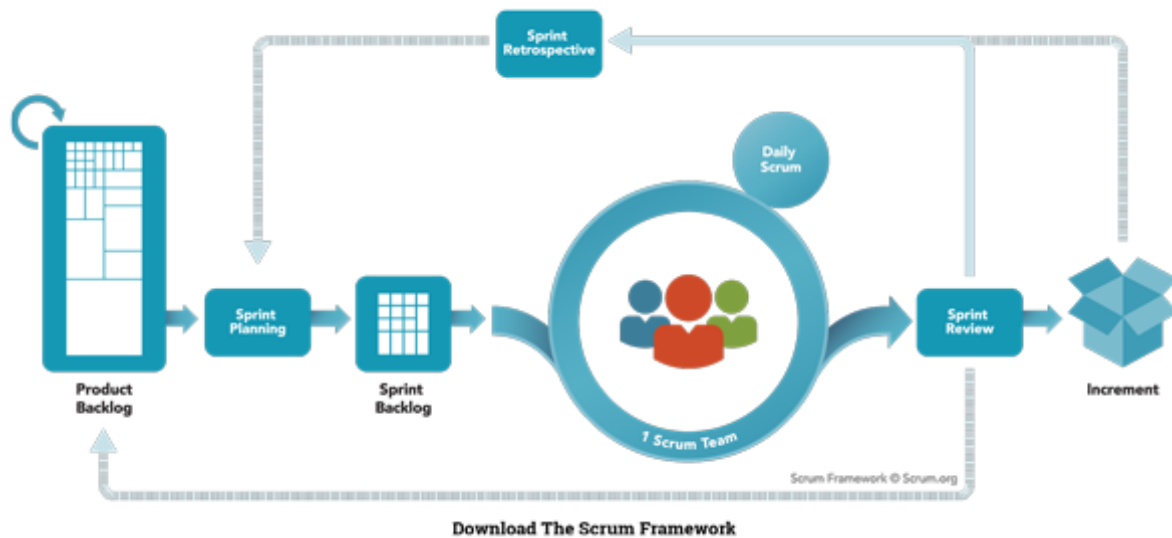


Figura 3.1: Diagrama proceso SCRUM. Fuente: https://s3.amazonaws.com/scrumorg-website-prod/drupal/inline-images/ScrumFramework_2000x1000.png

- **Sprint Backlog.** Su objetivo es seleccionar algunas de las funcionalidades escritas en el Product Backlog y marcar el tiempo en que van a ser desarrolladas. El primero aparece a continuación del Product Backlog.
- **Sprint Planning Meeting.** Antes de iniciar cada Spring (fase de desarrollo), se debe realizar un Sprint Planning Meeting, que consiste en definir plazos y procesos para el proyecto establecido.
- **Daily Scrum.** Se realizan diariamente en cada Sprint, en las que intentamos hacer una evaluación del desarrollo del proyecto para ver si se van cumpliendo los objetivos. El encargado de que todo vaya correctamente y de corregir los problemas es el **Scrum Master**, que se explicara más adelante.
- **Sprint Review.** Punto en el que se analiza lo desarrollado al final del Sprint, donde ya debería existir algo que mostrarle al cliente.
- **Sprint Retrospective.** Reunión para analizar los objetivos cumplidos y ver si existen fallos, y analizarlos para no cometer los mismos errores en el futuro. Este punto también sirve para ver la implementación de posibles mejoras

Durante el proceso es muy recomendable ordenar nuestro trabajo dentro de un tablero de tareas (**Scrum Taskboard**) para mejorar nuestra organización dentro del equipo y saber quién está realizando cada tarea, cuales quedan por realizar o están ya realizadas. Las categorías del tablero deben ser elegidas por el equipo.

Roles

A continuación veremos los diferentes roles dentro de un equipo que conforma la metodología SCRUM y sus respectivas responsabilidades.

- **Product Owner.** Es el líder general del proyecto, el encargado de velar por que se cumplan todos los objetivos.

- **Scrum Master.** Es el líder en cada reunión, la persona encargada de minimizar los problemas a lo largo del proyecto y dirigirlo por el camino correcto, por ello necesita tener los conocimientos de las herramientas usadas.
- **Scrum Team.** Son los encargados de desarrollar el proyecto y cumplir con los objetivos que le ha puesto el Product Owner.
- **Cliente.** Es una parte tan importante como involucrada en el proyecto, ya sea aportando nuevas ideas o comentarios sobre el desarrollo.

SCRUM aplicado al proyecto

Una vez que ya hemos explicado la metodología SCRUM, vamos a ver como la aplicamos al proyecto. Se ha seguido la metodología Scrum pero con algunas modificaciones, a pesar de que la generación de objetivos (*Requisitos*) y el diseño de la aplicación lo he realizado yo, como Scrum Team. Si he tenido un Product Owner encargado de que esos objetivos se cumplan en el proyecto. También he tenido a disposición un Scrum Master donde cada cierto tiempo se revisaba el trabajo realizado, se resolvían las dudas y se corregían los fallos cometidos.

Como podemos ver en la figura 3.2, hemos realizado un tablero de tareas, donde indicamos el trabajo a realizar, las que están siendo realizadas, en pruebas y las finalizadas.

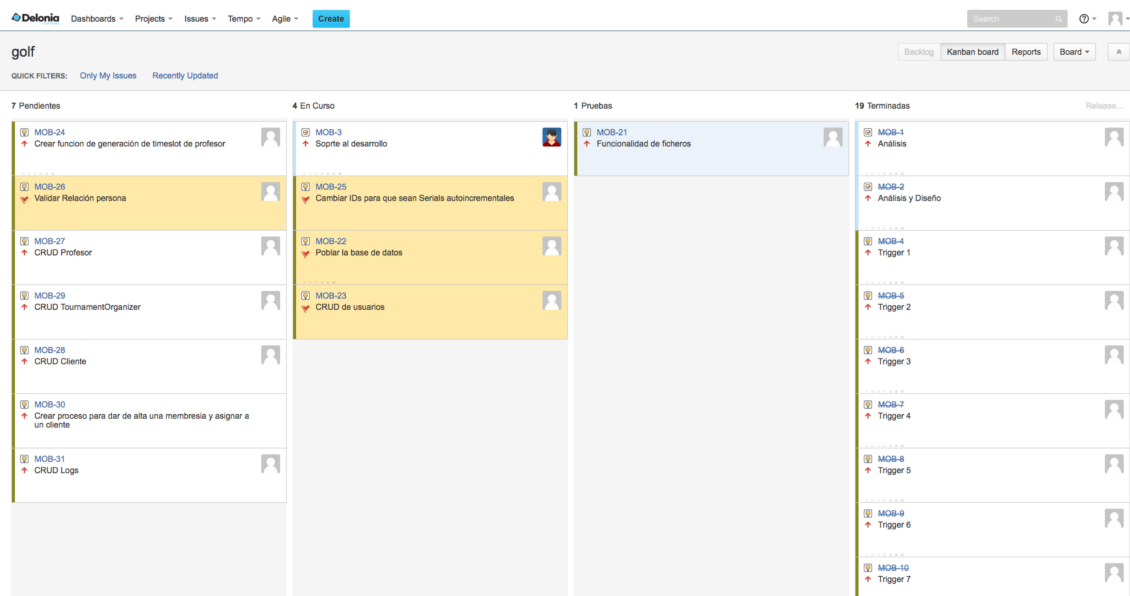


Figura 3.2: Tablero SCRUM.Fuente: Propia

3.2. Definición del proyecto

Es un sistema de gestión de clubs de golf para la empresa Delonía, la cual se encargará de distribuirlo a otras empresas (clubs), ayudándolas en la categoría de gestión de instalaciones, manteniendo la administración de los datos; en la categoría de gestión deportiva y operaciones, administrando su inventario, profesores y reservas; ayudando en una parte del apartado económico, controlando las transacciones económicas que se realicen con los clientes; e indirectamente ayudando en la gestión de empleados, ya que son los usuarios de la aplicación que les ayudará a mejorar su rendimiento en el trabajo.

El objetivo del proyecto es automatizar, mejorar y aumentar el rendimiento en la empresa.

3.3. Alcance del proyecto

El sistema está diseñado para la empresa Delonia, que se encargará de dar acceso a la aplicación, directamente o a través de canales de distribución, a los usuarios finales que serán clubs de golf y sus empleados.

A su vez está destinado para los empleados de cada club, que serán los usuarios más habituales en la aplicación y les ayudará en las distintas tareas de su día a día.

Los profesores también podrán hacer uso del sistema para administrar su perfil.

Los organizadores de torneos podrán entrar en el sistema para administrar su perfil y los torneos.

Por otro lado, Delonia, cuando distribuye la aplicación a través de canales de comercialización no se encarga de asegurar que el hardware instalado en los clubs cumpla los requisitos requeridos por la aplicación, si bien da unas pautas para la correcta ejecución de la aplicación.

3.4. Subsistemas del proyecto

Debido a lo grande que es el proyecto y que involucra diferentes tareas muy distintas, se ha dividido el proyecto en subsistemas para facilitar su desarrollo.

- **Subsistema de Gestión de Usuarios.** Básico en cualquier aplicación dirigida a un público específico. Consiste en administrar los usuarios de la aplicación con el objetivo de que estos puedan acceder a ella sin ningún problema y no tengan más acceso del asignado.
- **Subsistema de Gestión de Clientes y Socios.** Fundamental para la aplicación, se encargará de administrar los clientes de cada club, ya sean socios o no, y sus membresías, para que al final los empleados de los clubs puedan reservarles un timeslot en el campo.
- **Subsistema de Gestión de Campos y Productos.** Encargado de administrar los recorridos, sesiones, precios, ítems, pack, ect...
- **Subsistema de Gestión de Reservas.** Otro de los pilares de la aplicación, que incorpora el proceso con más frecuencia de uso, está encargado de crear una nueva reserva, modificarlas o cancelarlas.
- **Subsistema de Gestión de Torneos.** Subsistema secundario con respecto a otros más importantes, debido a su poca frecuencia de uso. Encargado de administrar los torneos, sus categorías y los jugadores suscritos a él.
- **Subsistema de Gestión de Profesores.** Otro de los subsistemas secundarios, debido a que la mayoría de los clientes no reservan clases cuando van a jugar. Se encarga de administrar los distintos procesos correspondientes a los profesores para que un cliente pueda reservar una clase.
- **Subsistema de Gestión de Clubs.** Subsistema sencillo, pero el más elemental de la aplicación, debido a que es un sistema multiempresa. Se encarga de administrar los clubs por parte de los usuarios del cliente.

- **Subsistema de Gestión de Facturas.** Imprescindible en cualquier aplicación que incorpore pagos, se encarga de gestionar las facturas que se generan en los demás subsistemas de la aplicación.
- **Subsistema de Comunicación con la BD.** El cliente posee un sistema de reservas y de suscripciones a torneos para los clientes, por lo que este subsistema se encargará de que este sistema pueda realizar operaciones en la base de datos a través de servicios web (**REST**).

A continuación, en la figura 3.3, vemos un pequeño esquema de cómo se relacionan los subsistemas de la aplicación. Debemos mencionar que los subsistemas de gestión de facturas y el de comunicación con la base de datos no están incluidos, ya que el de facturas se encuentra en cualquier transacción de dinero en el sistema y el de comunicación simplemente es un sistema para comunicarse con la base de datos.

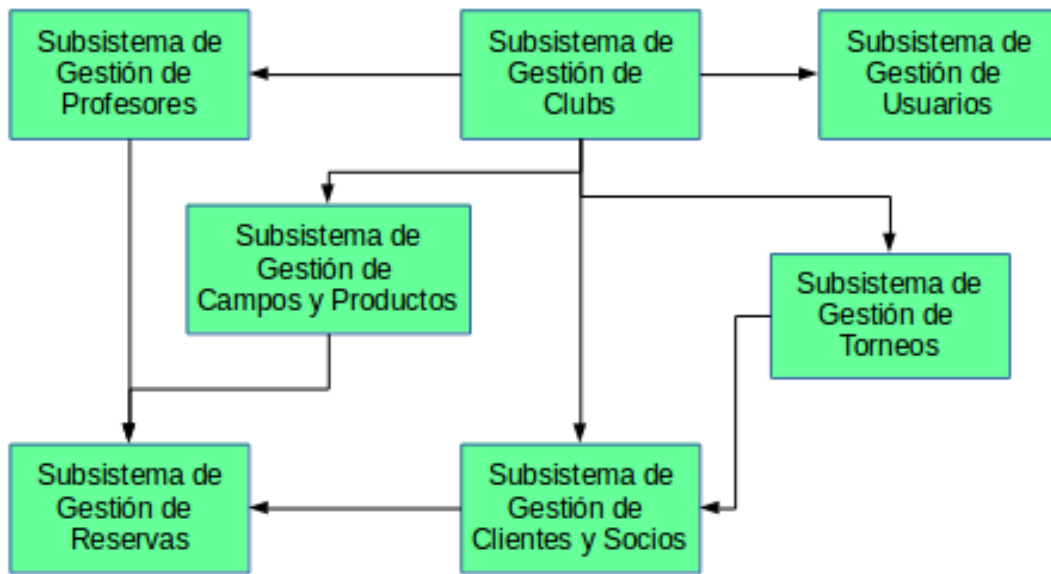


Figura 3.3: Diagrama de dependencia de los subsistemas. Fuente: Propia

4

Análisis

El **análisis** del proyecto nos sirve para examinar claramente cada uno de los puntos que engloban nuestro proyecto, de acuerdo a las necesidades del cliente. Es indispensable para poder desarrollar un buen software y para ello debemos seguir unas reglas.

4.1. Usuarios de la aplicación

La necesidad de diferenciar roles dentro de una aplicación se debe, sobre todo, a la seguridad de la aplicación. Es imprescindible marcar unos niveles de acceso, de manera que cada trabajador solo pueda utilizar y/o modificar la parte del programa, dependiendo del rol que se le ha asignado, evitando de esta manera, inseguridad, tanto en base de datos como en programación, que de otra manera harían mucho más vulnerable la aplicación.

La aplicación está dirigida principalmente a los trabajadores de cada club, pero como todo sistema, debe tener administradores y diferentes tipos de usuarios dentro de un mismo club. Por ello, a continuación vemos en la tabla 4.1 los diferentes roles y su definición.

Cuadro 4.1: Roles de Usuario

Administrador global	Es el rol con el permiso total de uso en la aplicación, pudiendo controlar varios clubs.
Administrador de club	Rol que tiene el permiso total sobre su club.
Empleado	Sera el usuario que más utilizará la aplicación, el encargado de administrar las reservas y los clientes del club en el que trabaja a diario.
Profesor	Usuario que solo podrá tener acceso a las partes que estén relacionados directamente con él.
Organizador de torneos	Usuario que solo tendrá acceso a sus datos personales y temas relacionados con los torneos del club donde este registrado.

Podemos decir que los roles administrador global, administrador de club y empleado están definidos como una jerarquía, es decir, todos los permisos que tiene el rol de empleado, están incluidos en los dos superiores, y a su vez, los roles de administrador de club, están incluidos en administrador global, aparte de los propios.

4.2. Análisis de requisitos

Nota: Cuando en la lista se dice administrar, engloba dar de alta, eliminar, modificar los datos, ver la información y buscar en una lista.

4.2.1. Requisitos funcionales

Subsistema de Gestión de Usuarios

- RF 1.** Un administrador global podrá administrar los usuarios.
- RF 2.** Un administrador de club podrá administrar los usuarios pertenecientes a su club.
- RF 3.** Un empleado de un club podrá administrar los usuarios con el rol de profesor u organizar de torneos dentro de su club.
- RF 4.** Cualquier usuario registrado previamente podrá loguearse en la aplicación.
- RF 5.** Cualquier usuario podrá cambiar su contraseña, y este recibirá un email con las instrucciones de recuperación y un código.
- RF 6.** Cualquier usuario que haya pedido el cambio de contraseña, podrá introducir el código de recuperación y cambiar su contraseña.

Subsistema de Gestión de Clientes y Socios

- RF 7.** Un empleado podrá administrar los clientes del club.
- RF 8.** Un empleado podrá ver el detalle del cliente.
- RF 9.** Un empleado podrá ver el listado de las membresías con las que está relacionado un cliente.
- RF 10.** Un empleado podrá ver los datos de cada una de las membresías anteriores.
- RF 11.** Un empleado podrá ver un listado de los clientes con derecho a uso de cada una de las membresías anteriores.
- RF 12.** Un empleado podrá ver el detalle de cada cliente miembro de cada una de las membresías anteriores.
- RF 13.** Un empleado podrá administrar las direcciones del cliente.
- RF 14.** Un empleado podrá ver un listado con las reservas del cliente.
- RF 15.** Un empleado podrá ver el detalle de cada reserva de un cliente.
- RF 16.** Un empleado podrá crear una membresía para un cliente, y se notificará a este vía SMS o email con los detalles.
- RF 17.** Un empleado podrá cambiar los permisos de envío de SMS y email del cliente por petición del mismo.
- RF 18.** Un empleado podrá ver un listado con las comunicaciones de un cliente.
- RF 19.** Un empleado podrá administrar las alertas de un cliente.

- RF 20.** Un empleado podrá marcar una alerta como “No mostrar más” o “Tratada”.
- RF 21.** Un empleado podrá bloquear o desbloquear a un cliente.
- RF 22.** Un empleado podrá subir ficheros para adjuntarlos al cliente.
- RF 23.** Un empleado podrá ver un listado con los jugadores con los que juega habitualmente un cliente.
- RF 24.** Un empleado podrá administrar los tipos de membresías.
- RF 25.** Un empleado podrá administrar las formas de pago de los tipos de membresías.

Subsistema de Gestión de Campos y Productos

- RF 26.** Un administrador de club podrá administrar los recorridos del club.
- RF 27.** Un administrador de club podrá administrar los hoyos del club.
- RF 28.** Un empleado podrá administrar los artículos.
- RF 29.** Un empleado podrá administrar las instancias de los artículos.
- RF 30.** Un empleado podrá administrar las sesiones de cada recorrido.
- RF 31.** Un empleado podrá administrar los precios de las reservas de las sesiones, dependiendo del tipo de socio, del canal de reserva, de si es individual o en grupo.
- RF 32.** El empleado podrá crear ofertas puntuales, promociones u ofertas de último minuto.
- RF 33.** El empleado podrá administrar packs con los Green Fees y los diferentes productos.
- RF 34.** El empleado podrá bloquear la instancia de un artículo por un tiempo o indefinidamente, o desbloquearlo.
- RF 35.** El empleado podrá establecer los días de cierre del club.
- RF 36.** El empleado podrá establecer el estado de un recorrido y el periodo que va a durar ese estado.
- RF 37.** El administrador del club puede añadir una foto de perfil del club.
- RF 38.** El empleado puede adjuntar fotos a un recorrido.

Subsistema de Gestión de Reservas

- RF 39.** Un empleado podrá realizar una reserva eligiendo:
 - Fecha de la reserva.
 - Tee time.
 - Los extras (Opcionales).
 - Los packs (Opcionales).
 - El número de jugadores.
 - Datos de los jugadores.
 - El recorrido.

- El profesor con el que quiere dar la clase (Opcional).

- RF 40.** Un empleado podrá asignar una instancia de un artículo a una reserva, cuando el cliente va a hacer efectiva la reserva en el club.
- RF 41.** En el caso de que la persona que hace la reserva no esté registrada en el sistema, un empleado podrá darla de alta como cliente.
- RF 42.** Un empleado podrá modificar el Tee time de una reserva, siempre que no esté pagada o el precio no cambie al mover la reserva.
- RF 43.** Un empleado podrá modificar los jugadores o extras de una reserva, siempre que no esté pagada o el precio no cambie al mover la reserva.
- RF 44.** Un empleado podrá cancelar una reserva.
- RF 45.** Un empleado podrá ver un listado de todas las reservas.
- RF 46.** Un empleado podrá ver los datos, jugadores y extras de una reserva.
- RF 47.** Un empleado podrá darle a un botón llamado “Que me espera hoy”, el cual mostrará un listado con todas las reservas para el día actual y sus respectivos jugadores.
- RF 48.** Un empleado podrá ver las tarifas más populares.
- RF 49.** Un empleado podrá hacer comparativas de los ingresos de distintos días.
- RF 50.** Un empleado podrá marcar una reserva como pagada.
- RF 51.** Un empleado podrá marcar una reserva como utilizada.
- RF 52.** Un empleado podrá indicar en un precio de sesión el importe a pagar si se cancela la reserva.
- RF 53.** Un empleado podrá generar los Tee time de un recorrido, indicando un periodo, el número máximo de jugadores y la diferencia de tiempo entre ellos.

Subsistema de Gestión de Torneos

- RF 54.** Un empleado podrá administrar los torneos.
- RF 55.** Un organizador de torneos podrá administrar los torneos.
- RF 56.** Un empleado podrá administrar los jugadores de torneos.
- RF 57.** Un organizador de torneos podrá administrar los jugadores de torneos.
- RF 58.** Un empleado podrá administrar los registros al torneo.
- RF 59.** Un organizador de torneos podrá administrar los registros al torneo.
- RF 60.** Un empleado podrá introducir los resultados del torneo de cada jugador.
- RF 61.** Un organizador de torneos podrá introducir los resultados del torneo de cada jugador.
- RF 62.** Un empleado podrá imprimir las etiquetas con el nombre, licencia y hándicap para pegarlas a las tarjetas de juego.
- RF 63.** Un organizador de torneos podrá imprimir las etiquetas con el nombre, licencia y hándicap para pegarlas a las tarjetas de juego.
- RF 64.** Un organizador de torneos podrá enviar el resultado del torneo a la Federación Española de Golf mediante el servicio web provisto por ellos.

Subsistema de Gestión de Profesores

- RF 65.** Un profesor podrá ver su agenda y posibilidad de reserva.
- RF 66.** Un profesor podrá administrar sus horarios de disponibilidad.
- RF 67.** Un profesor podrá administrar los precios en cada horario, dependiendo del día, la duración de la clase, categoría de la clase (individual o para grupos), para socios o no socios.
- RF 68.** Un empleado podrá administrar los profesores.
- RF 69.** El cliente y el profesor recibirán un email de confirmación donde se les informará de los detalles de la reserva.
- RF 70.** Un profesor podrá administrar sus idiomas.
- RF 71.** Un empleado podrá administrar los idiomas de un profesor.
- RF 72.** Un empleado podrá bloquear o desbloquear a un profesor.

Subsistema de Gestión de Facturas

- RF 73.** Un empleado podrá ver los detalles que van a incluirse en el fichero de domiciliación bancaria que se envía al banco con las cuotas de socio.
- RF 74.** Un empleado podrá exportar el fichero anterior a Excel.
- RF 75.** Un empleado podrá volver a lanzar un recibo no cobrado o rechazado.
- RF 76.** Un empleado podrá notificar al cliente por una cuota no pagada.
- RF 77.** Un empleado podrá generar una carta (PDF, basado en una plantilla predefinida) con los datos de la cuota del cliente.
- RF 78.** Un empleado podrá ver un listado con las cuotas del cliente.
- RF 79.** Un empleado podrá marcar una cuota como pagada.
- RF 80.** Un empleado podrá generar facturas (PDF, basado en una plantilla predefinida) para los clientes que lo deseen.
- RF 81.** Un profesor podrá generar los datos suficiente para generar su factura.
- RF 82.** Un administrador global podrá generar facturas (PDF, basado en una plantilla predefinida) de cada club.
- RF 83.** Un administrador de club podrá administrar los IVAs de su club.

Subsistema de Gestión de Clubs

- RF 84.** Un administrador global podrá administrar clubs.
- RF 85.** Un administrador global podrá bloquear o desbloquear a un club.
- RF 86.** Un administrador global podrá ver lo ingresos generados en un rango de tiempo.
- RF 87.** Un administrador global podrá ver la afluencia de clientes en rangos de tiempo.

Subsistema para la Comunicación con la BD

- RF 88.** Se permitirá crear, consultar, modificar o borrar en la base de datos a terceros a través de servicios web (REST). **Nota:** quedando pendiente definir las operaciones concretas a realizar.

4.2.2. Requisitos no funcionales

Rendimiento

- RNF 1.** Cualquier operación de la aplicación no debe tardar más de 1 segundo en completarse, a excepción de la obtención de estadísticas, informes y generación de Tee times.

Usabilidad

- RNF 2.** Debe tener una interfaz intuitiva, para que un nuevo empleado no tarde más de un día en aprender a utilizar el sistema.
- RNF 3.** Se debe acceder a los procesos de nueva reserva o hacer efectiva una reserva en menos de dos clics.

Seguridad

- RNF 4.** Un usuario de un club, no podrá acceder a información que no pertenezca a su club.

5

Diseño

El **diseño del software** es una de las etapas de las que está compuesto el ciclo de vida de un proyecto, posterior al análisis. Lo podemos entender como el objetivo de construir unos cimientos sobre los que se va a sostener el proyecto a lo largo de su ciclo de vida. Un buen diseño nos ayudara a realizar un proyecto de mejor calidad.

5.1. Diagramas de flujo

Un diagrama de flujo es la representación gráfica de un proceso. Representa cada aspecto del proceso únicamente mediante cajas y flechas.

En el anexo A podemos ver una serie de diagramas de flujo que se han realizado para entender mejor los procesos de nueva reserva, hacer una reserva efectiva, cancelar reserva y crear nueva membresía.

5.2. Maquetas

Se denomina maquetas a la construcción de un prototipo de las partes más complejas del sistema, para poder hacer ver de una mejor manera la interfaz al equipo, o mostrar al cliente como es nuestra idea de la interfaz. Con ello conseguimos que si al cliente no le gusta, podemos cambiarlo fácilmente, al contrario de tener que cambiar una pantalla realizada, debido a que el cambio de una pantalla hecha es mucho más complejo de modificar que una maqueta.

En el anexo B podemos ver una serie de maquetas, correspondientes al proceso de nueva reserva, que hemos realizado para que el cliente vea la idea que tenemos nosotros para la interfaz.

5.3. Base de datos

Entendemos como base de datos al conjunto de datos que pertenecen a un mismo concepto y sistemáticamente almacenado para posteriormente usarlo. En caso de una aplicación, se usan

las bases de datos para organizar y operar con los datos guardados en el servidor, en nuestro caso **PostgreSQL**.

Para realizar una aplicación de calidad y fácilmente mantenible, es imprescindible realizar una buena base de datos, ya que puede suponer más de la mitad de la calidad y responsable mayoritaria del rendimiento de la aplicación.

Nota: por motivos de confidencialidad, en la aplicación se han cambiado los nombres de las tablas y columnas, además de mostrar únicamente la estructura mínima necesaria para explicar los conceptos a continuación vistos.

5.3.1. Proceso

Para diseñar la base de datos, nos hemos encontrado con la necesidad de realizar tres fases, y en cada una, la realización de varios sprints, como hemos visto con el método SCRUM.

La primera fase ha consistido en la creación de una versión de la base de datos sin normalizar. Al final de cada sprint, se ha comprobado el diseño con el arquitecto de la empresa. Si consideraba que tenía recogidos todos los requisitos obtenidos en el análisis, daba por válida la versión de la base de datos.

La segunda fase consiste en normalizar la base de datos hasta la tercera forma normal, como explicaremos en el siguiente apartado. Para validar esta versión de la base de datos, se ha utilizado el mismo procedimiento que en la primera fase hasta obtener una base de datos consistente.

La tercera fase consiste en desnormalizar la base de datos para obtener un mejor rendimiento en las operaciones y unas consultas más sencillas. Para validar esta versión, que contemplaría una base de datos funcional y preparada para conectar con el backend, se ha seguido el mismo procedimiento que para las dos fases anteriores.

5.3.2. Normalización

Entendemos por normalización el proceso de organizar los datos en una base de datos, incluyendo la creación de tablas y de las relaciones entre ellas, basado en una serie de reglas para hacerla más flexible, eliminando la redundancia y las dependencias incoherentes. Los datos redundantes ocupan espacio en disco y dan problemas cuando debemos actualizarlos.

Existen una serie de reglas en la normalización de bases de datos, donde a cada una se la denomina “forma normal”. Hay multitud de conjuntos de reglas, pero la mayoría de las bases de datos están basadas en tres.

Si cumplimos el primer conjunto de reglas, decimos que la base de datos está en primera forma normal. Si cumplimos los tres primeros conjuntos de reglas, decimos que está en tercera forma normal. A continuación podemos ver en la tabla 5.1 el caso donde necesitamos guardar que tipo de membresía hemos escogido y quien tiene derecho a usarla [16].

Cuadro 5.1: Tabla de membresía sin normalizar

Membresía			
Código	Nombre-Membresía	Máximo-Miembros	Miembros
ADGT45	Familiar	4	Alberto, Alejandro, Manuel
YRDG67	Único	1	Enrique

Primera forma normal

Para cumplir la **primera forma normal** (1FN), los atributos deben incluir valores atómicos, y que el valor de cualquier atributo en una tupla debe ser un valor simple del dominio de ese atributo.

En este caso debemos modificar el atributo “Miembros” para convertirlo en un atributo atómico. Para ello, debemos crear un registro por cada valor en “Miembros”. Por último debemos modificar la clave primaria, que ahora se compondrá de dos atributos, “Código” y “Miembro”. Como vemos a continuación en las tablas 5.2.

Cuadro 5.2: Tabla de membresías en primera forma normal

Membresías			
<u>Código</u>	<u>Miembro</u>	Máximo-Miembros	Nombre-Membresía
ADGT45	Alberto	4	Familiar
ADGT45	Alejandro	4	Familiar
ADGT45	Manuel	4	Familiar
YRDG67	Enrique	1	Único

Segunda forma normal

La **segunda forma normal** (2FN) está basada en el concepto de dependencia funcional total. En el caso de que una clave primaria este compuesta por dos o más atributos, todo atributo que no pertenezca a la clave primaria, debe ser dependiente de cada uno de los atributos que componen la clave primaria.

En nuestro caso, podemos ver que “Máximo-Miembros” y “Nombre-Membresía” únicamente dependen del atributo primo “Código” y no de “Miembro”. Para pasar esta tabla a 2FN debemos crear una nueva tabla donde guardaremos los miembros relacionados con la membresías. Como vemos a continuación en las tablas 5.3 y 5.4.

Cuadro 5.3: Tabla de membresías en segunda forma normal

Membresías		
<u>Código</u>	Nombre-Membresía	Máximo-Miembros
ADGT45	Familiar	4
YRDG67	Único	1

Cuadro 5.4: Tabla de miembros de las membresías

MiembrosMembresía	
<u>Miembro</u>	<u>Código-Membresía</u>
Alberto	ADGT45
Alejandro	ADGT45
Manuel	ADGT45
Enrique	YRDG67

Tercera forma normal

La **tercera forma normal** (3FN) se basa en el concepto de dependencia transitiva. Este concepto se basa en que existe un conjunto de atributos que no son claves candidatas ni un

subconjunto de ninguna clave, es decir, si tenemos tres atributos, y el primero forma la clave primaria, el tercer atributo depende del segundo, y este a su vez depende del primero. Para que una tabla esté en 3FN debe eliminar este tipo de dependencias.

Como podemos ver en la tabla 5.3, el atributo “Máximo-Miembros” no depende de la clave primaria, si no que depende de “Nombre-Membresía”, ya que el número máximo que permite una membresía es dependiendo el tipo de membresía que hayas escogido. Para eliminar este caso de dependencia transitiva, debemos crear una tabla individual para guardar los tipos de membresías como vemos en la tabla 5.6 y eliminar el atributo “Máximo-Membresía” como vemos en la tabla 5.5.

Cuadro 5.5: Tabla de membresías en tercera forma normal

Membresías	
Código	Nombre-Membresía
ADGT45	Familiar
YRDG67	Único

Cuadro 5.6: Tabla de tipos de membresía

TipoDeMembresías	
Nombre	Máximo-Miembros
Familiar	4
Único	1

5.3.3. Desnormalización

Las reglas de normalización no consideran el rendimiento, por lo que en algunos casos es necesario desnormalizar la base de datos para mejorar el rendimiento de nuestra aplicación. Pero debemos dejar claro que para desnormalizar una base de datos, primero hemos tenido que normalizarla.

Como se ha visto en la normalización, hemos empezado con una tabla y hemos acabado teniendo hasta 3 tablas, lo que nos lleva a tener que volver a unir la información mediante consultas SQL, siempre que queremos conseguir información que se encuentra en varias tablas, por lo que en algunos casos puede darse el caso de demasiadas uniones. Por ello, normalizar una base de datos es una tarea imprescindible, pero a continuación debemos desnormalizarla para que el resultado sea un mejor rendimiento y unas consultas más sencillas.

A continuación, en la figura 5.1 vemos un ejemplo de desnormalización en la base de datos. Como vemos, la tabla “Sesión” tiene el identificador del club al que pertenece, con ello hemos conseguido que si queremos conocer el club al que pertenece la sesión solo debemos realizar la unión de dos tablas en la consulta, una unión menos que en el caso de no poseer el identificador de club en “Sesión”, en cuyo caso, nos haría hacer la unión de tres tablas.

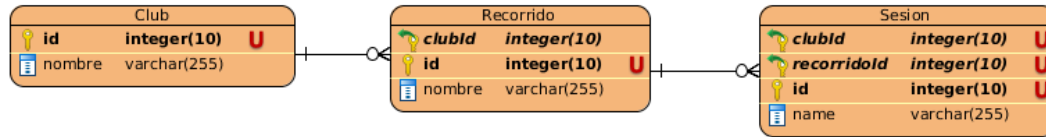


Figura 5.1: Desnormalización BD

5.3.4. Herencia

La herencia dentro de una base de datos es un intento de adaptar el modelo-entidad-relación al paradigma orientado a objetos. Se puede plasmar de dos formas, en una tabla única, donde existe una columna discriminadora, la cual contiene el valor que nos indicará a la clase que pertenece el registro, o creando una tabla por clase como en un diagrama de clases.

Para organizar los datos de las distintas personas que están involucradas en nuestra aplicación, ya sea haciendo uso de ella o no, hemos aplicado el modelo de herencia para mejorar la organización, como podemos ver en la figura 5.2. En este caso se ha escogido la segunda forma, una tabla por cada clase, debido que nuestro objetivo es que se diferenciaron e identificaran bien cada tipo de persona, además de poder guardar los datos únicos del tipo de persona en su tabla respectiva.

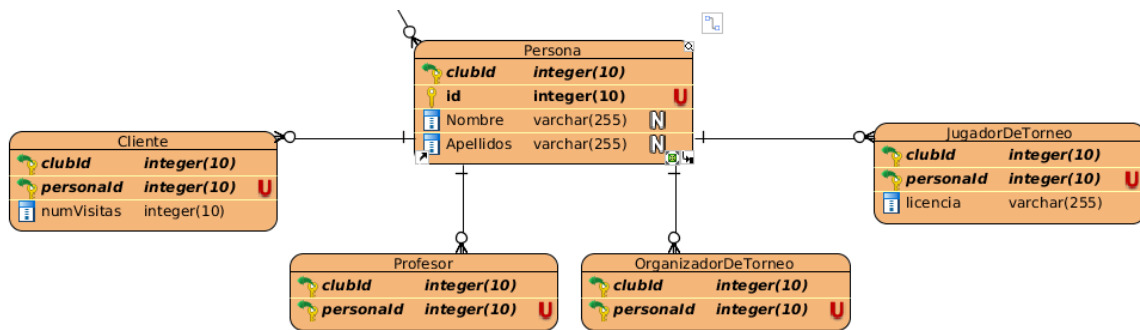


Figura 5.2: Herencia BD

6

Desarrollo

Una vez realizadas las fases de análisis y diseño, hemos visto que el tamaño del proyecto era más grande de lo esperado, por ello se ha decidido desarrollar en una primera fase la base de datos, una implementación inicial del backend y la arquitectura de las comunicaciones con el frontend para el TFG.

6.1. Generador de datos

Cualquier aplicación web debe ser probada de la forma más realista posible antes de ser entregada al cliente, por ello se ha realizado un pequeño generador de datos, realizado en Java, para poder poblar la base de datos, utilizando la librería DbUtils para conectarnos a la base de datos y poder realizar las consultas.

Para insertar los datos en una tabla, solo debemos crear una clase que extienda de la clase “TableFather” con el nombre de la tabla de base de datos, declarar solo y exclusivamente como atributos públicos las columnas que deseemos insertar, también debemos sobrescribir la función “initialization()” donde asignaremos los valores a las columnas, por último debemos llamar en el main a la función estática “doOperation(Connection conn, QueryRunner query, TableFather tabla, Integer num)”, el primer argumento es la conexión a la base de datos, el segundo argumento es la clase encargada de ejecutar las consultas, ambas clases son de la librería de DbUtils, el tercer argumento es la clase que nos hemos creado y el último argumento es el número de registros que se desea insertar.

Para entender mejor el generador de datos, en el anexo C se dispone del código y de un ejemplo de uso para una tabla.

Funciones

El programa dispone de una serie de funciones que nos ayudan a generar datos aleatorios para insertar en la base de datos. En la figura 6.1 podemos ver un pequeño diagrama de clases de la idea principal del generador.

- **create()**. Esta función es la encargada crear la consulta para insertar cada registro, solo debemos sobrescribirla cuando deseemos crear nuestra propia consulta.
- **createValues()**. Esta función devuelve los valores de las columnas que se van a insertar, entre paréntesis y separados por comas.
- **createColumns()**. Esta función devuelve una cadena con los nombres de las columnas entre paréntesis y separados por comas.
- **getNameTable()**. Si el nombre de la tabla es distinto al nombre de la clase, se debe sobrescribir esta función para que devuelva como String el nombre de la tabla.
- **random (Integer min, Integer max)**. Esta función devuelve un Integer entre dos números pasados por parámetro.
- **random (Long min, Long max)**. Esta función es igual a la anterior, pero manejando números más grandes, de carácter Long.
- **randomDate(Integer year)**. Esta función devuelve una objeto "Date", con una fecha aleatoria mayor que el año que hemos pasado como argumento.
- **randomDate(Date start, Date end)**. Esta función devuelve un objeto "Date" con una fecha aleatoria entre las dos fechas pasadas por parámetro.
- **randomTime(Integer start, Integer end)**. Esta función devuelve un objeto "Time" con una hora aleatoria entre las dos horas enteras pasadas por parámetro.
- **getDescripcion()**. Esta función devuelve una descripción con una probabilidad del 30 %.
- **getLocked()**. Esta función devuelve "true" un 10 % de las veces y el resto de las veces devuelve "false".
- **addDayToDate(Date date, int days)**. Esta función devuelve la fecha sumándole los días que se han pasado como parámetro.
- **addMonthToDate(Date date, int months)**. Esta función devuelve la fecha sumándole los meses que se han pasado como parámetro.
- **addYearToDate(Date date, int years)**. Esta función devuelve la fecha sumándole los años que se han pasado como parámetro.

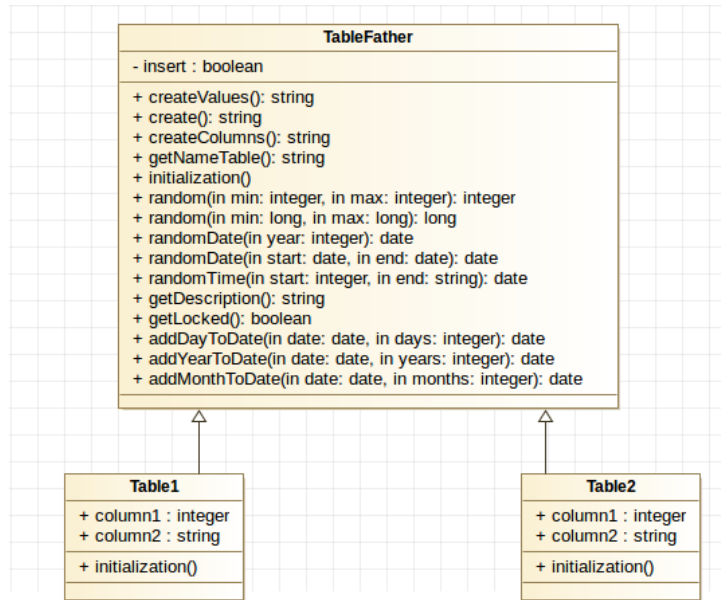


Figura 6.1: Diagrama de clases del generador de datos. Fuente: Propia

6.2. Tecnologías utilizadas

A continuación veremos las herramientas, frameworks y librerías utilizadas, para ayudarnos con el desarrollo del proyecto.

6.2.1. NetBeans IDE 8.2

Entorno de desarrollo libre. A pesar de que acepta multitud de lenguajes, está hecho principalmente para Java. Fue creado en Junio de 2000 por **MicroSystems**.

Este entorno de desarrollo lo he utilizado en el proyecto para implementar la parte del servidor y montar el servidor local con Apache para poder ir probando durante el desarrollo.

6.2.2. ReactJS

Es una librería de JavaScript de código abierto creada por Facebook. Utilizado en el frontend, se caracteriza por crear un Virtual DOM y en vez de renderizar el DOM, consigue cambiar únicamente la parte donde existan modificaciones [17].

Antiguamente, las aplicaciones web dejaban la mayoría del trabajo de procesamiento al servidor, debido a la poca potencia de procesador de los clientes. Durante los últimos años esta idea ha ido cambiando, los clientes han mejorado mucho más su capacidad de procesamiento que los servidores, por ello, librerías como ReactJS son una gran herramienta a utilizar, ya que nos ayudan a delegar la gran mayoría de las tareas de procesamiento en el cliente, liberando al servidor de carga de trabajo.



Figura 6.2: Logo ReactJS. Fuente: http://frontendlabs.io/wp-content/uploads/2016/10/react_js.png

ReactJS vs Angular 2

ReactJS y Angular 2 son dos herramientas para el desarrollo del frontend, usadas para crear páginas *Single Page Applications*, pero con diferentes características, las cuales veremos en la tabla comparativa 6.1 [18].

Cuadro 6.1: Comparación de Angular 2 y ReactJS

	Angular 2	ReactJS
Autor	Google	Facebook
Tamaño	764k	151k
DOM	Regular DOM	Virtual DOM
Diseño de código	JS en HTML	Centrado en JS
Curva de aprendizaje	Media	Baja
Debugging	Necesidad de plugins	Necesidad de plugins
Binding	2 vías	Una dirección
Templating	TypeScript	JSX
Móvil	Ionic Framework	React Native

6.2.3. Uaithne

Framework para facilitar el desarrollo del backend de un proyecto, que permita crecer sin que cada vez resulte más complejo de mantener. Se completa con un generador de código que permite generar todo el código necesario para la implementación del backend, así como la lógica de acceso a base de datos. Está basado en los patrones de diseño **Command** (Orden), **Chain of Responsibility** (Cadena de responsabilidades) y **Strategy** (Estrategia), los cuales dan una estructura para construir el backend, permitiendo incorporar muchas ventajas de la Programación Orientada a Aspectos [19].

Podemos asemejarlo a un juego de lego, donde se disponen de piezas altamente intercambiables que se colocan hasta conseguir el resultado deseado.

6.2.4. Visual Paradigm

Visual Paradigm es una herramienta para el diseño de distintos diagramas, como pueden ser diagramas UML o diagramas entidad-relación. En nuestro caso hemos utilizado Visual Paradigm para diseñar la base de datos con un diagrama entidad-relación, no solo creando las tablas y sus relaciones, sino también los constraints que debe tener la base de datos, los datos por defecto de las columnas y los ids auto-generados.

Una de las grandes ventajas que tiene esta herramienta, es que nos puede generar un código para crear la base de datos a partir del diagrama y empezar desde ahí.

6.2.5. pgAdmin

pgAdmin es la plataforma de administración y desarrollo de código abierto más popular y con más características para PostgreSQL, la base de datos de código abierto más avanzada del mundo. Puede ejecutarse en Linux, Unix, Mac OS y en Windows para administrar bases de datos en PostgreSQL 9.2 o superior [20].

En nuestro caso es la herramienta que hemos utilizado para crear la base de datos y realizar las pruebas sobre ella.

6.3. Arquitectura de la aplicación

La arquitectura de la aplicación está basada en una arquitectura web, donde el cliente hace una petición de la página web a través de su navegador y el servidor le responde enviándole la página HTML, como vemos en la figura 6.3, y partir de ahí empieza una comunicación cliente-servidor basada en el protocolo clásico **RPC** (Remote Procedure Call) a través de envíos de cadenas Json que ejecutan las operaciones. El servidor está montado en **Apache Tomcat**.

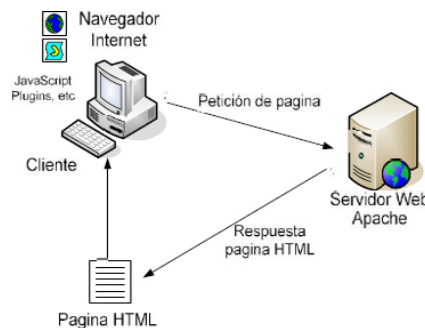


Figura 6.3: Arquitectura web. Fuente: http://imagenes.mailxmail.com/cursos/imagenes/6/4/arquitectura-de-base-de-datos-para-la-web_22846_5_1.jpg

6.3.1. Apache Tomcat

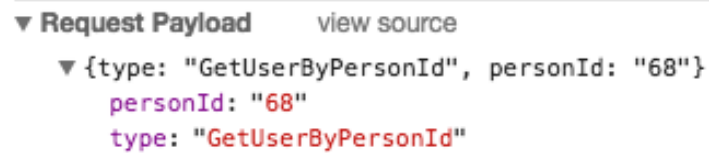
Apache Tomcat es una implementación de código abierto de Java Servlet, JavaServer Pages, Java Expression Language y Java WebSocket. Esta tecnología está desarrollada por **Apache Software Foundation** (ASF). Está desarrollado bajo la licencia de **Apache License version 2**.

Tomcat es un contenedor web donde se presenta a menudo con el servidor web de Apache, aunque puede funcionar como un servidor web por sí mismo. En la actualidad, es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

6.3.2. Comunicación cliente-servidor

Como ya hemos mencionado la comunicación está basada en el modelo clásico de comunicación RPC y Json con el servidor.

El cliente crea un Json con la operación que desea ejecutar y los parámetros que debe de llevar esa operación, cómo podemos ver en la figura 6.4, una vez serializado el Json, el cliente lo envía al servidor a través de una petición *POST* del protocolo HTTP, y esto es posible gracias al uso de AJAX, que nos permite realizar comunicación con el servidor sin recargar la página.



```

▼ Request Payload    view source
  ▼ {type: "GetUserByPersonId", personId: "68"}
    personId: "68"
    type: "GetUserByPersonId"
  
```

Figura 6.4: Petición Json. Fuente: Propia

El servidor recibe el Json y lo deserializa en un objeto Java, a continuación ejecuta la operación indicada en el Json con sus respectivos parámetros. Una vez ejecutada la operación, el servidor crea otro Json con la respuesta, cómo podemos ver en la figura 6.5, ya sea un objeto o una lista de objetos, y se lo envía al cliente.



x	Headers	Preview	Response	Cookies	Timing
1			}}',		
2			{"clubId":2,"login":"Andrea68","personId":68,"userType":2}		

Figura 6.5: Respuesta Json. Fuente: propia

Por último, el cliente recibe la respuesta y deserializa el Json en un objeto JavaScript para ser usado en el navegador.

6.4. Base de datos

Para desarrollar la estructura básica de la base de datos, únicamente hemos generado el script “.sql”, y lo hemos ejecutado en pgAdmin para crear los cimientos de la base de datos.

Además se han creado una serie de trigger para controlar los temas de borrado de los registros. Con ello, cuando borremos algún registro de la base de datos y necesitemos borrar también algunos registros de otra tabla que dependan de ellos, los trigger serán los encargados de borrarlos, eliminando la necesidad de hacerlo en el backend.

Uno de los trigger más importantes que tenemos es el encargado de borrar una persona, ya que cuando el usuario desee eliminarla, se debe cambiar toda la información personal del sujeto, todas sus direcciones y adjuntos deben de ser borrados, también su usuario si lo tuviera; si es un profesor se deben inactivar sus horarios; si es un cliente se borrarán sus memberships y membershipMember y si es un jugador de torneos se actualizará su licencia.

7

Pruebas Realizadas y Resultados

7.1. Base de datos

Para probar el correcto funcionamiento de la base de datos, primero se ha poblado con datos ficticios, creados con el pequeño generador de datos que hemos visto antes. Una vez que tenemos la base de datos llena de información empezamos a realizar las pruebas.

Primero comprobamos que los constraints que hemos definido en la base de datos funcionan correctamente, para ello hemos realizado inserciones con la intención de que nos dé como resultado un error por las restricciones de los constraints.

A continuación debemos comprobar las relaciones entre las tablas, para ello hemos realizado consultas a la base de datos uniendo las tablas y comprobando que la información que deseamos ver es la que hemos extraído.

Después debemos comprobar el correcto funcionamiento de los trigger, esto es una tarea sencilla, ya que solo debemos ejecutar la consulta que hace que los trigger se activen y comprobar que la tarea del trigger se ha ejecutado correctamente y de la forma que se ha previsto.

Por ultimo debemos comprobar el rendimiento. Para ello primero es muy importante cargar la base de datos con la información que preveremos que vamos a manejar en una situación real en el futuro. Una vez tengamos datos suficientes en las tablas, solo debemos realizar las consultas que tenemos previsto que el backend realizara sobre la base de datos y medir el tiempo desde que empieza la consulta hasta que termina. Por lo general, una consulta que tarda más de 350 milisegundos es una consulta lenta y se deben tomar medidas para corregir esa pérdida de rendimiento. Existen algunos casos de excepción donde es inevitable que la consulta pase de los 350 milisegundos, como puede ser la necesidad de traernos una gran cantidad de información. A continuación, en las figuras 7.1 y 7.2, vemos el ejemplo de la diferencia de tiempo de una consulta sin desnormalizar las tablas y una consulta con las tablas desnormalizadas para obtener la misma información. Queremos obtener el nombre del club al que pertenece la sesión con id igual a 5.

```
EXPLAIN SELECT c.nombre FROM Sesión s JOIN Recorrido c on (s.recorridoId=c.id) JOIN club cl on (c.clubId=cl.id)
WHERE s.id=5;
```

	QUERY PLAN text
1	Nested Loop (cost=8.45..16.10 rows=1 width=16)
2	-> Hash Join (cost=8.30..15.89 rows=1 width=20)
3	Hash Cond: (c.id = s.courseid)
4	-> Seq Scan on course c (cost=0.00..6.60 rows=260 width=24)
5	-> Hash (cost=8.29..8.29 rows=1 width=4)
6	-> Index Scan using season_id_key on season s (cost=0.28..8.29 rows=1 width=4)
7	Index Cond: (id = 5)
8	-> Index Only Scan using club_pkey on club cl (cost=0.14..0.20 rows=1 width=4)
9	Index Cond: (id = c.clubid)

Figura 7.1: Consulta sin desnormalizar. Fuente: Propia

```
EXPLAIN SELECT c.nombre FROM Sesión s JOIN club c on (s.clubId=c.id)
WHERE s.id=5;
```

	QUERY PLAN text
1	Hash Join (cost=8.30..10.04 rows=1 width=13)
2	Hash Cond: (c.id = s.clubid)
3	-> Seq Scan on club c (cost=0.00..1.53 rows=53 width=17)
4	-> Hash (cost=8.29..8.29 rows=1 width=4)
5	-> Index Scan using season_id_key on season s (cost=0.28..8.29 rows=1 width=4)
6	Index Cond: (id = 5)

Figura 7.2: Consulta con desnormalización. Fuente: Propia

Como podemos ver la consulta sin desnormalizar tiene que realizar un mayor número de comparaciones y por lo tanto un mayor coste de trabajo que la consulta contra la base de datos desnormalizada. Además, podemos ver que la consulta es más sencilla cuando la base de datos esta desnormalizada.

7.2. Pruebas de verificación

Las pruebas de verificación son las encargadas de comprobar que el producto se está construyendo correctamente y que los requisitos marcados se están cumpliendo. Para comprobar su correcto funcionamiento hemos realizado varios tipos de pruebas.

7.2.1. Pruebas backend

Pruebas de caja blanca

Las pruebas de caja blanca se han realizado dentro de cada operación. Como la mayoría de operaciones son únicamente una consulta a base de datos, se han realizado directamente las pruebas de caja negra, que explicaremos a continuación. Pero para las operaciones más complejas se ha comprobado el correcto funcionamiento de cada línea de código, que cada bucle se ejecuta el número de veces deseado y que cada consulta a la base de datos nos dé el resultado esperado.

Pruebas de caja negra

Las pruebas de caja negra consisten en hacer pruebas sobre las funciones, en este caso sobre las operaciones, sin conocer su código interior, únicamente introduciendo los parámetros y comprobando la salida.

Por ejemplo, para conseguir la información de una persona solo debemos llamar a la operación "SelectPersonById" pasando el id de la persona como parámetro, y comprobando que la información que obtenemos es la correcta.

8

Conclusiones y trabajo futuro

El Trabajo de Fin de Grado ha consistido en la realización de un sistema de gestión de clubs de golf para la empresa Delonia, la cual dará acceso a los distintos clubs a la aplicación.

Durante la elaboración del análisis y diseño, podíamos ver que el proyecto es muy grande, por ello se ha tomado la decisión de realizar únicamente el desarrollo de una primera fase para el TFG, que consiste en la implementación de la base de datos, el desarrollo inicial del backend de la aplicación y la arquitectura de las comunicaciones con el frontend.

Una vez puesto en producción el sistema, Delonia podrá empezar a dar acceso a nuestra aplicación, y a su vez, los clubs podrán hacer uso de ella y aprovechar todas las ventajas que les proporciona para su gestión.

Como ya vimos anteriormente, durante la elaboración del proyecto hemos seguido la metodología de SCRUM, la cual nos ha ayudado a ver los errores y corregirlos en fases más tempranas.

Durante las fases de análisis y diseño hemos aplicado los conocimientos adquiridos en la escuela acerca de Ingeniería del Software, como puede ser, el análisis de requisitos o la creación de maquetas, todo ello para rectificar errores con el cliente antes de empezar el desarrollo, ya que cuanto antes identifiquemos los fallos, menos cambios se deberán realizar en el proyecto.

Los conocimientos adquiridos durante el proyecto, y que servirán para un futuro, han sido estos:

- El descubrimiento de comunicaciones con cadenas Json, a través de peticiones “POST” mediante el protocolo HTTP.
- La ampliación de los conocimientos de Java para construir un servidor.
- Aprender a desarrollar con la librería de ReactJS, la cual está muy extendida en el mundo del desarrollo web.
- Aprender a desarrollar el backend de una aplicación con el framework Uaithne.
- El descubrimiento de LaTeX como sistema para crear documentos.

Como se ha comentado durante la memoria, solamente se ha implementado una primera de fase del desarrollo. Por este motivo debe completarse el desarrollo de la segunda fase, es este

caso la interfaz, para poder poner el sistema en producción. Esta segunda fase se compone de diferentes iteraciones.

- **Primera iteración.** Donde se desarrollará la interfaz de usuario de los subsistemas de gestión de usuarios, campos y productos, clubs y clientes/socios.
- **Segunda iteración.** Donde se implementará la interfaz de usuario de los subsistemas de gestión de profesores y de reservas. Solamente desarrollamos dos subsistemas en esta iteración, ya que el de reservas es el más complicado y grande de todos ellos.
- **Tercera iteración.** Por último, se desarrollará la interfaz de usuario de los subsistemas de gestión de torneos, facturas y de comunicación con la base de datos.

Una vez que el proyecto esté terminado se pondrá en producción y empezará la fase de mantenimiento, donde nosotros debemos asegurarnos de que el sistema funcione correctamente y realizar los distintos cambios que el cliente vaya proponiendo a lo largo de su ciclo de vida. De aquí la necesidad de haber diseñado una aplicación fácil de mantener.

Para el futuro, debemos de mantener la posibilidad de ampliar el sistema, siempre por petición del cliente, para que sean los propios clientes los que realicen las reservas y mantengan su perfil.

Glosario de acrónimos

- **Sistema ERP:** *Enterprise resource planning*. Es un sistema de información que integra y maneja los negocios asociados con las operaciones de una compañía en la producción de bienes o servicios.
- **TFG:** Trabajo de Fin de Grado
- **Green Fee:** Precio estipulado para poder jugar en un campo de golf.
- **Tee time:** Nombre que recibe la hora para empezar a jugar.
- **Hándicap:** Valoración otorgada a cada jugador por la federación de cada país.
- **SASS:** *Syntactically Awesome Stylesheets*. Extensión del lenguaje CSS.
- **LESS:** Extensión del lenguaje CSS.
- **BD:** Base de Datos
- **RF:** Requisito funcional.
- **RNF:** Requisito no funcional.
- **DOM:** *Document Object Model* Interfaz de programación para representar documentos HTML, XHTML y XML.
- **REST:** *Representational state transfer*. Forma para proporcionar interoperabilidad entre sistemas informáticos en Internet.
- **HTTP:** *Hypertext Transfer Protocol*. Protocolo de comunicación para la World Wide Web
- **RPC:** *Remote Procedure Call*. Cuando un ordenador ejecuta código en otra máquina remota.
- **AJAX:** *Asynchronous JavaScript And XML*. Técnica que permite la transferencia asíncrona de datos entre un navegador y un servidor.
- **SQL:** *Structured Query Language*. Lenguaje que da acceso a un sistema de gestión de bases de datos relacionales.

Bibliografía

- [1] Portal web golfindustria. <http://golfindustria.es/asesorias-gestion-clubs/>. Accedido: 14-05-2017.
- [2] anónimo. Historia del golf. https://es.wikipedia.org/wiki/Historia_del_golf. Accedido: 11-05-2017.
- [3] <https://www.golfspain-mastergolf.com/>. Accedido: 10-04-2017.
- [4] <http://www.mastelhospitality.com/golf-software-solutions/>. Accedido: 10-04-2017.
- [5] Juan Diego Gauchat. El gran libro de html5, css3 y javascript. pages 18–41, 2012.
- [6] Juan Diego Gauchat. El gran libro de html5, css3 y javascript. pages 66–83, 2012.
- [7] Dan Iel. Sass y less : El presente de css. <http://maquetando.com/sass-y-less-el-presente-de-css>. Accedido: 22-05-2017.
- [8] ¡bienvenido a la referencia de lenguaje de unity! <https://docs.unity3d.com/es/current/ScriptReference/index.html>. Accedido: 22-05-2017.
- [9] Anónimo. ¿qué es php? <http://php.net/manual/es/intro-what-is.php>. Accedido: 19-04-2017.
- [10] Anónimo. Php. <https://es.wikipedia.org/wiki/PHP>. Accedido: 19-04-2017.
- [11] Anónimo. Oracle database. https://es.wikipedia.org/wiki/Oracle_Database. Accedido: 19-04-2017.
- [12] Anónimo. Microsoft sql server. https://es.wikipedia.org/wiki/Microsoft_SQL_Server#Desventajas. Accedido: 20-04-2017.
- [13] rafaelpma. Sobre postgresql. http://www.postgresql.org.es/sobre_postgresql. Accedido: 20-04-2017.
- [14] <http://www.rae.es/>. Accedido: 21-04-2017.
- [15] OK HOSTING. Metodologías del desarrollo de software. https://okhosting.com/blog/metodologias-del-desarrollo-de-software/#Metodologia_Scrum. Accedido: 21-04-2017.
- [16] Ramiz Elmasri y Shamkant B. Navathe. Fundamentos de sistemas de bases de datos, 5ª edición. pages 18–41, 2007.
- [17] reicek. Qué es y cómo funciona reactjs. <https://platzi.com/blog/intro-react-js/>. Accedido: 23-04-2017.

- [18] Shannon Duncan. Angular vs react : A side-by-side comparison. <https://www.pluralsight.com/guides/front-end-javascript/angular-vs-react-a-side-by-side-comparison>. Accedido: 23-05-2017.
- [19] Juan Luis Paz Rojas. Manual de uaithne. Manual de uso personal Delonia Software. Accedido: 23-04-2017.
- [20] <https://www.pgadmin.org/>. Accedido: 19-05-2017.



Diagramas de flujo

A continuación se verán los diagramas de flujo de los procesos más complejos del sistema, y por tanto los que necesitan ser vistos de una forma más sencilla para llegar a comprenderlos bien y realizarlos con la mejor calidad posible. Para representar las acciones de usuario hemos utilizado el color verde y el azul para las del sistema.

A.0.1. Proceso de nueva reserva

Este proceso se realizará cuando un empleado vaya a crear una nueva reserva por petición de un cliente.

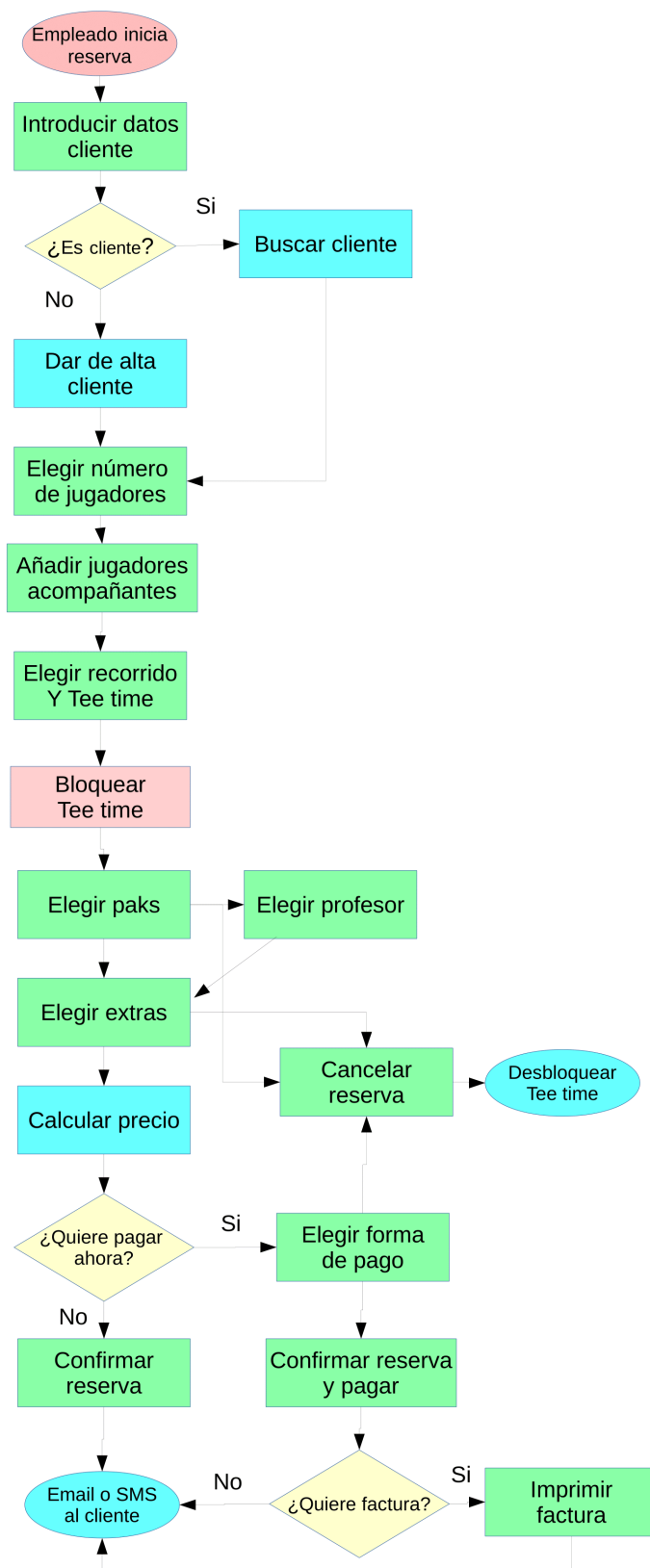


Figura A.1: Diagrama de flujo del proceso de nueva reserva

A.0.2. Proceso para hacer una reserva efectiva

Cuando un cliente llega al club para jugar, y tiene una reserva realizada con antelación, el empleado deberá hacer las oportunas gestiones y confirmar la asistencia del cliente.

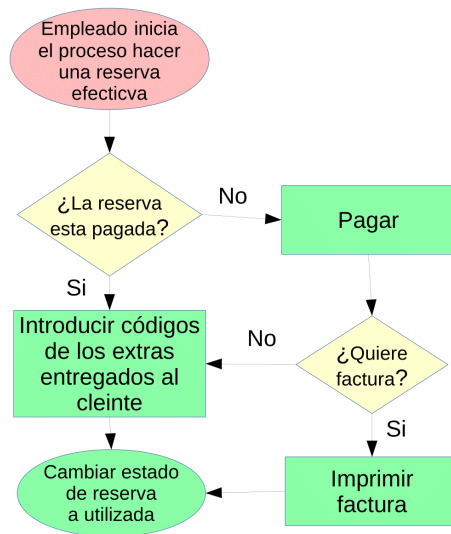


Figura A.2: Diagrama de flujo del proceso para hacer efectiva una reserva

A.0.3. Proceso de cancelación de reserva

Cuando un cliente desea cancelar una reserva que ha realizado anteriormente, un empleado debe iniciar el proceso de cancelación de la misma.

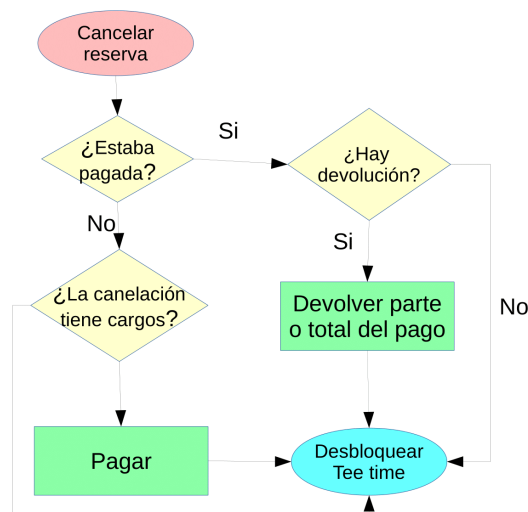


Figura A.3: Diagrama de flujo del proceso de cancelar reserva

A.0.4. Proceso de nueva membresía

Cuando un cliente, este o no registrado en el sistema, desea contratar una nueva membresía, un empleado debe realizar el proceso de nueva membresía paso a paso.

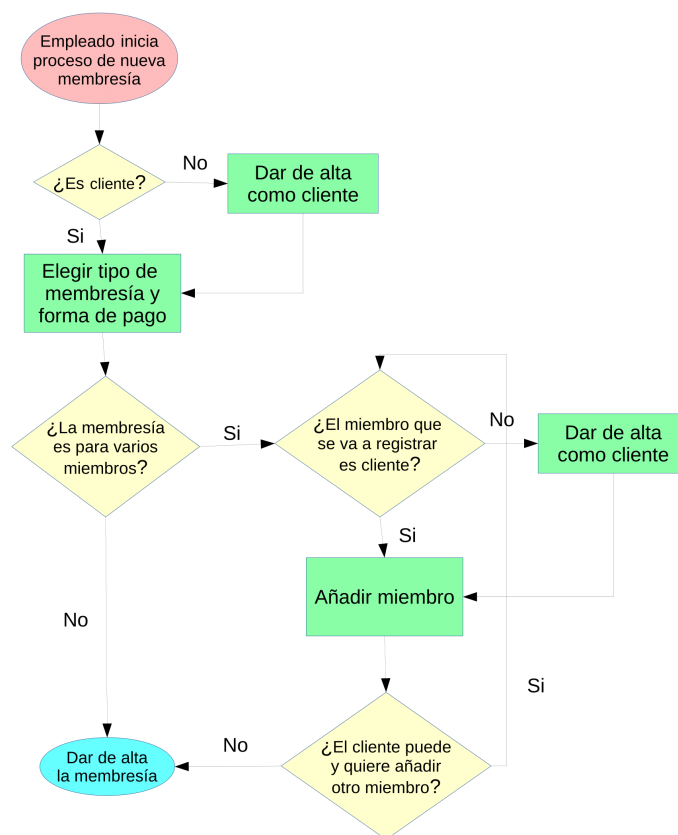


Figura A.4: Diagrama de flujo del proceso de crear una nueva membresía

B

Maquetas

B.0.1. Proceso de nueva reserva

En nuestro caso hemos querido presentar al cliente las maquetas del proceso de nueva reserva, el más complejo de ellos, dividido en tres etapas: la primera es un formulario para introducir los datos del cliente y acompañantes, la segunda donde se da la opción de elegir el Tee time y en la última puede elegir los packs, extras y si quiere un profesor.

1ª Etapa

Reserva			
Cliente			
DNI: 12345678	Nombre: Alberto	Apellidos: Perez Garcia	Tipo de abono: Abono familiar <input checked="" type="checkbox"/>
Acompañantes 3			
Acompañante 1			
DNI: 987654321	Nombre: Gonzalo	Apellidos: Perez Garcia	Tipo de abono: Abono familiar <input checked="" type="checkbox"/>
Acompañante 2			
DNI: 743698210	Nombre: Roberto	Apellidos: Perez Torralba	Tipo de abono: Sin abono <input type="checkbox"/>
Acompañante 3			
DNI: 246789012	Nombre: Maria	Apellidos: Gonzalez Fernandez	Tipo de abono: Sin abono <input type="checkbox"/>
<input type="button" value="Cancelar"/> <input type="button" value="Siguiente"/>			

Figura B.1: Primera etapa del proceso de nueva reserva

Como vemos en la figura B.1, lo primero que debe hacer el usuario es introducir los datos del cliente que va a realizar la reserva. Si pulsas “Enter” el sistema buscara al cliente, si los datos coinciden con solo uno, el sistema rellenará automáticamente los campos y pondrá un “tick” verde como el que vemos en la imagen, el aspa indica que no se ha encontrado o aún no se ha pulsado la tecla “Enter”.

A continuación el usuario deberá elegir el número de acompañantes y rellenar los datos de estos, al igual que introdujimos los datos del cliente que hace la reserva.

Por ultimo debemos pulsar el botón “Siguiente” para continuar a la segunda fase. En caso de que el cliente no se encuentre en el sistema, el usuario debe añadirlo antes de continuar, ya que si no se ha encontrado en la base de datos, el sistema no dejará pasar a la siguiente fase.

2ª Etapa

The screenshot shows a web application interface for golf reservations. On the left is a sidebar with navigation links: Home, Profesores, Reservas, Gestión de torneos, Gestión de clientes, Gestión de tienda, and Facturación socios. The main area is titled 'Reserva' and contains three sections:

- Cliente:** A form with fields for Name (Alberto), Surname (Perez Gracia), DNI (12345567K), Type of membership (Abono familiar), Number of companions (3), and Amount (30 €).
- Calendario:** A calendar for July 2016. The 11th is highlighted with a lock icon, indicating it is closed for reservations.
- Tee Times:** A table showing available tee times from 8:00 to 19:00. Each row includes a time slot, two dropdown menus for 'Recorrido 1' and 'Recorrido 2', and a final column showing the number of available spots. For example, at 12:00, there are 9 spots available for the selected route.

At the bottom right of the table are buttons for 'Cancelar' and 'Siguiente'.

Figura B.2: Segunda etapa del proceso de nueva reserva

Como vemos en la figura B.2, arriba a la izquierda podemos ver un recuadro con los datos del cliente que está realizando la reserva y el precio que deberá pagar al final de ella por el Tee time.

Abajo a la izquierda vemos un calendario donde visualizamos los días que está cerrado. Si pinchamos en un día o nos movemos con las fechas, el recuadro de la derecha se actualizará al día indicado.

En el recuadro de la derecha podemos ver los Tee times del día que hemos escogido en el calendario y los recorridos escogidos en el combo que vemos arriba a la derecha. Para elegir un Tee time solo debemos pinchar en un hueco libre, que son los recuadros verdes con el número de plazas en ese Tee time como vemos en la maqueta. Para mover una reserva que ya se había realizado antes, solo debemos arrastrarla al hueco deseado. La última columna de la tabla indica el número de artículos que más se usan y están disponibles en ese momento.

Una vez que ya hemos elegido el horario, solo debemos pulsar el botón “Siguiente” para pasar a la última etapa, y el sistema reservará automáticamente el Tee time durante 10 minutos para que en el transcurso de terminar el proceso no se pueda reservar desde otro acceso el Tee time elegido.

3ª Etapa

Como en la etapa de antes podemos ver los datos del cliente que está haciendo la reserva en el recuadro de arriba a la izquierda.

Reserva

Ciente

Nombre	Apellidos	DNI	Tipo de abono
Alberto	Perez Gracia	12345567K	Abono familiar

Packs

Pack	Green Fee	Carrito	Price
Pack 1	Green Fee + Carrito electrico		25 € +
Pack 2	Green Fee + Juego completo de palos de golf		40 € +
Pack 3	Green Fee + Carrito electrico + Comida		30 € +

Carrito 79 €

Recorrido 2 12:40 0 €

Artículos Principales

Nombre	Carrito	Price
Carrito electrico		15 € +
Carrito Manual		10 € +
Carrito Manual		5 € +
Juego completo de palos de golf		20 € +

Extras

Nombre	Descripción	Price
Desayuno	Zumo de naranja + Cafe + Toastadas	3,5 € +
Hiero 9	Loft: 42° - 44°, Lie: 62,5°	5 € +
Hiero 8	Loft: 38° - 40°, Lie: 62°	4,5 € +

Profesores

Nombre	Precio/h	Idiomas	¿Le ha dado clase?
Roberto Perez Torroja	20 €	Castellano, Inglés	Si
José Fernandez Rodriguez	15 €	Castellano	No
Laura Albarca Diaz	25 €	Castellano, Inglés, Francés, Alemanía	No

Carrito

Nombre	Cantidad	Precio Total
Pack 1	1	25 €
Juego com...	2	40 €
Desayuno	4	14 €

Buttons: Cancelar, Siguiente

Figura B.3: Tercera etapa sin elección del profesor del proceso de nueva reserva

En la figura B.3, en el recuadro de la derecha vemos los packs que podemos elegir, solo debemos pulsar en el símbolo del más. Justo abajo podemos elegir los artículos más demandados igual que los packs, con el símbolo del más.

El siguiente recuadro de abajo muestra una lista con el resto de los artículos, podemos elegir uno pulsando el símbolo más de la fila.

A la derecha podemos ver el carrito de la compra, viendo arriba lo que cuesta el Green Fee que hemos escogido y abajo una lista con los packs, artículos y/o profesor que hemos elegido. El precio final lo podemos ver en el título del recuadro.

Reserva

Ciente

Nombre	Apellidos	DNI	Tipo de abono
Alberto	Perez Gracia	12345567K	Abono familiar

Packs

Pack	Green Fee	Carrito	Price
Pack 1	Green Fee + Carrito electrico		25 € +
Pack 2	Green Fee + Juego completo de palos de golf		40 € +
Pack 3	Green Fee + Carrito electrico + Comida		30 € +

Carrito 79 €

Recorrido 2 12:40 0 €

Artículos Principales

Nombre	Carrito	Price
Carrito electrico		15 € +
Carrito Manual		10 € +
Carrito Manual		5 € +
Juego completo de palos de golf		20 € +

Extras

Nombre	Descripción	Price
Desayuno	Zumo de naranja + Cafe + Toastadas	3,5 € +
Hiero 9	Loft: 42° - 44°, Lie: 62,5°	5 € +
Hiero 8	Loft: 38° - 40°, Lie: 62°	4,5 € +

Profesores

Nombre	Precio/h	Idiomas	¿Le ha dado clase?
Roberto Perez Torroja	20 €	Castellano, Inglés	Si
José Fernandez Rodriguez	15 €	Castellano	No
Laura Albarca Diaz	25 €	Castellano, Inglés, Francés, Alemanía	No

Buttons: No Quiero Profesor, Cancelar, Siguiente

Figura B.4: Tercera etapa con elección del profesor del proceso de nueva reserva

Como podemos ver abajo a la izquierda de la figura B.3, tenemos un botón “Quiero profesor” donde al pinchar en él abriremos una lista con los profesores disponibles, como vemos en la figura B.4. Para contratarlo solo debemos hacer doble clic en él.

Hemos decidido crear un botón para seleccionar al profesor debido a que reservar una clase es un acción muy poco común comparada con hacer una reserva sin clase, con lo que así conseguimos más espacio para el listado de artículos.



Generador de datos

C.1. Código

Como veremos a continuación, el generador de datos dispone de dos **Main**. El llamado “MainPrueba” se utiliza para probar que nuestra clase inserte bien los registros y no tengamos ningún error al ejecutar. El segundo Main se llama “Main” y lo podemos entender como el Main principal, una vez que estamos seguros de que nuestra clase funciona perfectamente, debemos preparar este Main y ejecutarlo.

La única diferencia entre los dos es que el Main principal crea seis hilos para que la generación de datos se realice en el menor tiempo posible, siendo este número de hilos el ideal para un procesador de 8 núcleos.

MainPrueba.java

```
package com.delonia.generadordatos;

import org.apache.commons.dbutils.QueryRunner;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.commons.dbutils.DbUtils;

public class MainPrueba {
    public static final Logger logger =
        Logger.getLogger(Main.class.getName());

    public synchronized static void main(String[] args) throws
        SQLException, IllegalArgumentException,
```

```
IllegalAccessException, InterruptedException {
    Connection conn = null;
    String url = "";
    String driver = "org.postgresql.Driver";
    String user = "";
    String pwd = "";

    try {
        DbUtils.loadDriver(driver);
        conn = DriverManager.getConnection(url, user, pwd);
        QueryRunner query = new QueryRunner();
        /*
        Insertar codigo
        */
    } catch (SQLException ex) {
        logger.log(Level.SEVERE, "Imposible conectarse a la
        BBDD", ex);
    } catch (IllegalArgumentException ex) {
        Logger.getLogger(Thread1.class.getName()).log(Level.SEVERE,
        null, ex);
    } finally {
        try {
            DbUtils.closeQuietly(conn);
        } catch (Throwable ex) {
            logger.log(Level.SEVERE, "Error al cerrar la
            conexion origen", ex);
        }
    }
}

}
```

Main.java

```
package com.delonia.generadordatos;

import org.apache.commons.dbutils.QueryRunner;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.logging.Logger;

public class Main {
    public static final Logger logger =
        Logger.getLogger(Main.class.getName());
}
```

```
public synchronized static void main(String[] args) throws
SQLException, IllegalArgumentException,
IllegalAccessException, InterruptedException {
    Thread1 t1 = new Thread1();
    Thread2 t2 = new Thread2();
    Thread2 t3 = new Thread2();
    Thread2 t4 = new Thread2();
    Thread2 t5 = new Thread2();
    Thread2 t6 = new Thread2();
    Thread2 t7 = new Thread2();

    t1.setThread(t2, t3, t4, t5, t6, t7);
    t2.setThread(t1, 2);
    t3.setThread(t1, 3);
    t4.setThread(t1, 4);
    t5.setThread(t1, 5);
    t6.setThread(t1, 6);
    t7.setThread(t1, 7);

    Thread thread1 = new Thread( t1, "thread1" );
    Thread thread2 = new Thread( t2, "thread2" );
    Thread thread3 = new Thread( t3, "thread3" );
    Thread thread4 = new Thread( t4, "thread4" );
    Thread thread5 = new Thread( t5, "thread5" );
    Thread thread6 = new Thread( t6, "thread6" );
    Thread thread7 = new Thread( t7, "thread7" );

    thread1.start();
    thread2.start();
    thread3.start();
    thread4.start();
    thread5.start();
    thread6.start();
    thread7.start();

    thread1.join();
    thread2.join();
    thread3.join();
    thread4.join();
    thread5.join();
    thread6.join();
    thread7.join();
}

public static void doOperation(Connection connOrig,
QueryRunner query, TableFather tabla, Integer num) throws
SQLException, IllegalArgumentException,
IllegalAccessException{
    int i;
```

```
        for (i=0; i< num; i++){
            tabla.initialization();
            query.update(connOrig, tabla.create());
        }
    }
}
```

Thread1.java

```
package com.delonia.generadordatos;

import static com.delonia.generadordatos.Main.logger;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.commons.dbutils.DbUtils;
import org.apache.commons.dbutils.QueryRunner;

public class Thread1 extends Thread{
    private Thread2 thread2;
    private Thread2 thread3;
    private Thread2 thread4;
    private Thread2 thread5;
    private Thread2 thread6;
    private Thread2 thread7;
    private Integer num;

    public void setThread (Thread2 thread2, Thread2 thread3,
        Thread2 thread4, Thread2 thread5, Thread2 thread6, Thread2
        thread7){
        this.thread2 = thread2;
        this.thread3 = thread3;
        this.thread4 = thread4;
        this.thread5 = thread5;
        this.thread6 = thread6;
        this.thread7 = thread7;
    }

    public synchronized void run() {
        Connection conn = null;
        String url = "";
        String driver = "org.postgresql.Driver";
        String user = "";
        String pwd = "";
    }
}
```

```
try {
    //Conexion TIM
    DbUtils.loadDriver(driver);
    conn = DriverManager.getConnection(url, user, pwd);

    QueryRunner query = new QueryRunner();

    /*Numero de registros que deseamos insertar*/
    this.slipThread(10000000);

    this.wait();
    this.wait();
    this.wait();
    this.wait();
    this.wait();
    this.wait();

} catch (SQLException ex) {
    logger.log(Level.SEVERE, "Imposible conectarse a la
        BBDD", ex);
} catch (IllegalArgumentException | InterruptedException
    ex) {
    Logger.getLogger(Thread1.class.getName()).log(Level.SEVERE,
        null, ex);
} finally {
    try {
        DbUtils.closeQuietly(conn);
    } catch (Throwable ex) {
        logger.log(Level.SEVERE, "Error al cerrar la
            conexion origen", ex);
    }
}

}

public synchronized void slipThread (Integer num){
    Integer cociente = num/6;
    this.thread7.setNum(cociente);
    this.thread2.setNum(cociente);
    this.thread3.setNum(cociente);
    this.thread4.setNum(cociente);
    this.thread5.setNum(cociente);
    this.thread6.setNum(cociente);
    this.thread2.notifyTh();
    this.thread3.notifyTh();
    this.thread4.notifyTh();
    this.thread5.notifyTh();
    this.thread6.notifyTh();
    this.thread7.notifyTh();
}
```

```
        public synchronized void notifyTh () {
            this.notify();
        }
    }
}
```

Thread2.java

```
package com.delonia.generadordatos;

import static com.delonia.generadordatos.Main.logger;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import lombok.Data;
import org.apache.commons.dbutils.DbUtils;
import org.apache.commons.dbutils.QueryRunner;

public class Thread2 extends Thread {
    private Thread1 thread1;
    private Integer idThread;
    private Integer num;

    public void setThread (Thread1 thread1, Integer id) {
        this.thread1 = thread1;
        this.idThread = id;
    }

    public synchronized void run() {
        Connection conn = null;
        String url = "";
        String driver = "org.postgresql.Driver";
        String user = "";
        String pwd = "";

        try {
            DbUtils.loadDriver(driver);
            conn = DriverManager.getConnection(url, user, pwd);

            QueryRunner query = new QueryRunner();

            this.wait();
            /*
            Insertar codigo
            */
        } catch (Exception e) {
            logger.error(e.getMessage());
        }
    }
}
```

```
        */
        this.thread1.notifyTh();

    } catch (SQLException ex) {
        logger.log(Level.SEVERE, "Imposible conectarse a la
        BBDD", ex);
    } catch (IllegalArgumentException ex) {
        Logger.getLogger(Thread1.class.getName()).log(Level.SEVERE,
        null, ex);
    } catch (InterruptedException ex) {
        Logger.getLogger(Thread2.class.getName()).log(Level.SEVERE,
        null, ex);
    } finally {
        try {
            DbUtils.closeQuietly(conn);
        } catch (Throwable ex) {
            logger.log(Level.SEVERE, "Error al cerrar la
            conexion origen", ex);
        }
    }
}

public synchronized void notifyTh (){
    this.notify();
}

public synchronized void waitTh () throws
InterruptedException{
    this.wait();
}
}
```

TableFather.java

```
package com.delonia.generadordatos;

import java.lang.reflect.Field;
import java.lang.reflect.Type;
import java.sql.Date;
import java.sql.Time;
import java.sql.Timestamp;
import java.util.Calendar;
import java.util.Random;

public class TableFather {
    private Boolean insert = true;
```

```
public String createValues () throws
IllegalArgumentException, IllegalAccessException{
    Field[] values = this.getClass().getFields();
    String res = "(";
    for(Field value : values){
        Type o= value.getGenericType();
        if
            (value.getName().equals("id")||(value.getName().equals("number")
            ||(!value.getName().equals("vatId") &&
            value.getName().contains("Id") &&
            o.getTypeName().equals("java.lang.String"))){
            res = res + value.get(this) + ",";
        } else if (value.get(this) == null){
            res = res + "NULL" + ",";
        } else if (o.getTypeName().equals("java.sql.Date")){
            Calendar cal = Calendar.getInstance();
            Date date = (Date) value.get(this);
            cal.setTime(date);
            res = res + "TO_DATE(' " + this.getDate(cal)+ " ',
            'dd/mm/yyyy')" + ",";
        } else if
            (o.getTypeName().equals("java.sql.Timestamp")){
            Calendar cal = Calendar.getInstance();
            Timestamp date = (Timestamp) value.get(this);
            cal.setTime(date);
            res = res + "TO_DATE(' " + this.getDate(cal) + " ',
            'dd/mm/yyyy')" + ",";
        } else if
            (o.getTypeName().equals("java.lang.Double")){
            Double num = (Double) value.get(this);
            String numS = num.toString();
            res = res + numS.substring(0, numS.indexOf("."))
            + "." + numS.substring(numS.indexOf(".")+1) +
            ",";
        } else if
            (o.getTypeName().equals("java.lang.String")){
            String cad = (String) value.get(this);
            String print = cad;
            if (cad.contains("'")){
                Integer first = 0;
                print = cad.substring(0, cad.indexOf("'")) +
                ",'";
                String aux =
                cad.substring(cad.indexOf("'")+1);
                while (aux.contains("'")){
                    print = print + aux.substring(0,
                    aux.indexOf("'")+ ",'";
                    aux = aux.substring(aux.indexOf("'")+1);
                }
            }
        }
    }
}
```



```
        print = print + aux;
    }
    res = res + "'" + print + "'" + ",";
} else if
    (o.getTypeName().equals("java.lang.Boolean") ||
    o.getTypeName().equals("java.lang.Integer")){
    res = res + value.get(this) + ",";
} else {
    res = res + "'" + value.get(this) + "'" + ",";
}

}
res = res.substring(0, res.length()-1);
return res + ")";
}

public String create() throws IllegalAccessException{
    String namesTable = this.createColumns();
    String valueTable = this.createValues();
    return "INSERT INTO " + this.getNameTable()+ " " +
        namesTable + " values" + valueTable;
}

public String createColumns() {
    Field[] values = this.getClass().getFields();
    String res = "(";
    for(Field value : values){
        switch (value.getName()) {
            case "order":
                res = res + "\"" + value.getName() + "\", ";
                break;
            default:
                res = res + value.getName() + ", ";
                break;
        }
    }

    return res.substring(0, res.length()-1) + ")";
}

public String getNameTable() {
    return this.getClass().getSimpleName();
}

public void initialization(){

public Timestamp dateToTimestamp (Date date){
    if (date==null){
        return null;
    }
    Calendar cal = Calendar.getInstance();
```

```
        cal.setTime(date);
        cal.set(Calendar.MILLISECOND, 0);
        return new java.sql.Timestamp(cal.getTimeInMillis());
    }

    public Integer random (Integer min, Integer max){
        Double num = (Math.random()*(max-min)+0.5)+min;
        if (num.intValue()>max){
            num--;
        }
        return num.intValue();
    }

    public long random (Long min, Long max){
        Double num = (Math.random()*(max-min)+0.5)+min;
        if (num.intValue()>max){
            num--;
        }
        return num.longValue();
    }

    public Date randomDate(Integer year){
        int numero = 0;
        Random aleatorio;
        aleatorio = new Random();

        Calendar unaFecha = Calendar.getInstance();
        unaFecha.set (aleatorio.nextInt(10)+year,
            aleatorio.nextInt(12)+1, aleatorio.nextInt(30)+1);
        return new Date(unaFecha.getTimeInMillis());
    }

    public Date randomDate(Date start, Date end){
        long aleatorio = this.random(start.getTime(),
            end.getTime());
        return new Date(aleatorio);
    }

    public Time randomTime(Integer start, Integer end){
        Integer hour = this.random(start, end);
        Integer minutes = this.random(1, 59);
        return new Time(hour, minutes, 0);
    }

    public String getDescripcion(){
        if (Math.random()<0.3){
            return "Soy una description";
        }
        return null;
    }
}
```

```
public Boolean getLocked(){
    return Math.random() < 0.1;
}

public Date addDayToDate(Date date, int days){

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    calendar.add(Calendar.DAY_OF_YEAR, days);
    return new Date(calendar.getTime().getTime());
}

public Date addMonthToDate(Date date, int months){

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    calendar.add(Calendar.MONTH, months);
    return new Date(calendar.getTime().getTime());
}

public Date addYearToDate(Date date, int years){

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(date);
    calendar.add(Calendar.YEAR, years);
    return new Date(calendar.getTime().getTime());
}

public String getDate(Calendar cal){
    return cal.get(Calendar.DAY_OF_MONTH) + "/" +
        (cal.get(Calendar.MONTH)+1) + "/" +
        cal.get(Calendar.YEAR);
}
}
```

Count.java

```
package com.delonia.generadordatos;

import java.sql.Date;
import java.sql.Time;

public class Count {
    public Integer entero;
    public Integer entero2;
    public Integer entero3;
    public String cadena;
```

```
public String cadena2;
public String cadena3;
public Date fecha;
public Boolean bool;
public Time hora1;
public Time hora2;
public Double decimal;
}
```

Enumerates.java

```
package com.delonia.generadordatos.destination;

import java.util.ArrayList;
import java.util.Arrays;

public class Enumerates {
    public static ArrayList<String> nameBoys = new
        ArrayList<>(Arrays.asList("Roberto", "Pablo", "Ernesto",
            "Guillermo", "Sergio", "Alvaro", "Raul", "Alejandro"));
    public static ArrayList<String> nameGirls = new
        ArrayList<>(Arrays.asList("Carolina", "Laura", "Paloma",
            "Leticia", "Andrea", "Lucia", "Maria"));
    public static ArrayList<String> lastnames = new
        ArrayList<>(Arrays.asList("Perez", "Blazquez", "Pardo",
            "Carmona", "Rodriguez", "Albarca", "Molina", "Diez",
            "Gracia", "Fernandez", "Hernandez", "Fuentes", "Beltran"));
    public static ArrayList<String> companies = new
        ArrayList<>(Arrays.asList("Empresa 1", "Empresa 2",
            "Empresa 3"));
    public static ArrayList<String> cities = new
        ArrayList<>(Arrays.asList("Madrid", "Toledo", "Burgos",
            "Barcelona", "Sevilla", "Albacete", "Murcia", "Valencia",
            "Cadiz"));
    public static ArrayList<String> colors = new
        ArrayList<>(Arrays.asList("Azul", "Rojo", "Amarillo",
            "Blanco", "Negro", "Verde"));
    public static ArrayList<String> items = new
        ArrayList<>(Arrays.asList("Palo", "Carrito", "Juego de
            Palos"));
    public static ArrayList<String> languages = new
        ArrayList<>(Arrays.asList("es", "en", "de", "fr", "it",
            "ar", "zh"));
}
```

C.2. Ejemplo

A continuación mostraré un ejemplo de la utilización del generador de datos con la siguiente tabla de la base de datos.






Persona		
	clubId	integer(10)
	id	integer(10) U
	nombre	varchar(255)
	apellidos	varchar(255) N
	genero	numeric(2, 0) N
	handicap	integer(10) N

Figura C.1: Tabla Persona.

Persona

Listing C.1: Persona.java

```
package com.delonia.generadordatos.destination;

import com.delonia.generadordatos.Count;
import com.delonia.generadordatos.TableFather;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
import lombok.Data;
import org.apache.commons.dbutils.QueryRunner;
import org.apache.commons.dbutils.handlers.BeanHandler;

@Data
public class Persona extends TableFather{
    private Connection conn;
    public Integer clubId;
    public String nombre;
    public String apellidos;
    public Integer genero;
    public Integer handicap;

    public Persona (Connection conn){
        this.conn = conn;
    }

    @Override
    public void initialization(){
        try {
            QueryRunner query = new QueryRunner();
            Count clubMax = (Count) query.query(conn, "SELECT
                max(id) as entero FROM club", new
                BeanHandler(Count.class));
            this.clubId=this.random(1, clubMax.entero);
        }
    }
}
```

```
        this.genero=this.random(1, 2);
        this.initName();
        this.handicap=this.random(10, 100);
    } catch (SQLException ex) {
        Logger.getLogger(Persona.class.getName()).log(Level.SEVERE,
            null, ex);
    }
}

private void initName () {
    if (this.genero==1){
        Integer aleatorio = this.random(0,
            Enumerates.nameBoys.size()-1);
        this.nombre = Enumerates.nameBoys.get(aleatorio);
    } else {
        Integer aleatorio = this.random(0,
            Enumerates.nameGirls.size()-1);
        this.nombre = Enumerates.nameGirls.get(aleatorio);
    }

    Integer aleatorio = this.random(0,
        Enumerates.lastnames.size()-1);
    this.apellidos = Enumerates.lastnames.get(aleatorio);
    this.apellidos = this.apellidos + " " +
        Enumerates.lastnames.get(aleatorio);
}
}
```

Si queremos ejecutar el Main de prueba deberemos insertar el siguiente código en la clase MainPrueba en el lugar donde se indica.

```
Persona persona = new Persona(conn);
Main.doOperation(conn, query, persona, 100);
```

Si queremos ejecutar el Main principal debemos indicar cuantos registros deseamos insertar en la clase Thread1, e insertar el siguiente código en la clase Thread2 en el lugar donde se indica.

```
Persona persona = new Persona(conn);
Main.doOperation(conn, query, persona, num);
```